Joint
Transportation
Research
Program

JTRP

FHWA/IN/JHRP-96/21 -01

Final Report
Volume I (Introduction & Appendix 1)

THREE DIMENSIONAL FINITE ELEMENT PROGRAMS FOR PAVEMENT ANALYSIS

T. Nilaward
C. Shih
T. White
E. Ting

Indiana
Department
of Transportation

Purdue
University

FINAL REPORT

FHWA/IN/JHRP-96/21

THREE DIMENSIONAL FINITE ELEMENT PROGRAMS FOR PAVEMENT ANALYSIS

Volume I (Introduction & Appendix 1)

by

Tatsana Nilaward
and
Chiang Shih
Research Assistants

and

Thomas D. White
and
Edward C.Ting
Research Engineers

Purdue University
Department of Civil Engineering

Joint Transportation Research Program
Project No: C-36-67FF
File No: 5-8-32

Prepared in Cooperation with the
Indiana Department of Transportation and
the U.S. Department of Transportation
Federal Highway Administration

Purdue University
West Lafayette, IN 47907
January 1998

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/IN/JHRP-96/21		

4. Title and Subtitle	5. Report Date
Three Dimensional Finite Element Program for Pavement Analysis	January 1998
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
T. Nilaward, C. Shih, T. White, and E. Ting	FHWA/IN/JHRP-96/21

9. Performing Organization Name and Address	10. Work Unit No.
Joint Transportation Research Program 1284 Civil Engineering Building Purdue University West Lafayette, Indiana 47907-1284	11. Contract or Grant No. HPR-2073

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Indiana Department of Transportation State Office Building 100 North Senate Avenue Indianapolis, IN 46204	Final Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared in cooperation with the Indiana Department of Transportation and Federal Highway Administration.

16. Abstract

A three dimensional finite element program is developed for the analysis of pavement systems. An explicit approach of the finite element analysis is adopted. This approach results in a vector formulation of the equation of motion. Large displacement is considered through the use of a co-rotational approach which considers small deformations and large rotations of the elements. For the convenience of application a two dimensional finite element program is also developed. An eight-node isoparametric solid element is used for the three dimensional analysis, and a four-node element for the two dimensional analysis.

Loading conditions are verified for static ramp and step loadings, sinusoidal loadings, prescribed ground acceleration input, and pulse input. The material library is verified for linear elastic materials, elastic-plastic materials with Mises or Drucker-Prager criteria and assuming associated or non-associated flow rules, and a viscoelastic material of Maxwell type. Three hardening rules are implemented, namely the kinematical hardening, isotropic hardening, and the mixed type. Available analytical data and comparison studies by using ANSYS serve as the basis for the verification.

17. Key Words	18. Distribution Statement
pavement, computer, software finite element, material model.	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	461 (3 volumes)	

Form DOT F 1700.7 (8-69)

TABLE OF CONTENTS

Page

LIST OF FIGURES

Implementation Report

In general, highway and airfield pavements consist of a system of layers resting on a foundation. In situ, the layers can be nonlinear and vary in thickness, homogeneity and isotropy. They have boundary and interaction conditions that significantly affect their response and performance. Loading can be static, moving or dynamic. In spite of these conditions the pavement industry has, by use, largely defined rational pavement analysis and design as a linear or stepwise linear, layered elastic problem. In doing so, the layers are assumed to be linear, homogeneous and isotropic. The layers are assumed to extend to infinity and interactions are limited. Loading is taken to be static.

The major reason given for continued emphasis on use of the layered elastic model is that it is easy to use. There are various layered elastic programs that run on personal computers (PC). Even on a PC, solutions are obtained in a fraction of a second. Models based on the finite element method of analysis can more realistically represent complex pavement problems. Industry has resisted use of such models, citing complicated input and need for workstation or mainframe computers. Also, computing times are large. In fact, PCs have become much more powerful and alternate algorithms for conducting FEM analyses have been developed.

A study was conducted that involved development of a FEM algorithm that would run on a PC. The algorithm is an explicit solution and involves a vector formulation representing the equations of motion. Both two- and three-dimensional versions of the program have been developed. The program is in a modular format. Initial libraries of properties have been provided for the load and material modules, respectively. Additions can be made to the libraries as needed. Loads can range from static to dynamic. Material

models include linear and nonlinear elastic, viscoelastic and plastic or a combination. Both the loads and material models have been verified.

The three-dimensional program has been used to predict deflection response of jointed concrete pavement subjected to an impulse load from a falling weight deflectometer (FWD). The predicted response was in close agreement with in situ measurements. Other example problems have been demonstrated showing that the programs can be utilized for the general analyses of pavement systems (concrete and asphalt) subjected to transient dynamic loading histories.

The study advisory committee recommended that the report and programs be accepted and therefore satisfy project goals. As a preliminary of the committee arriving at this recommendation, a presentation was made on options for developing input files and displaying results. The committee decided that general implementation would be greatly enhanced by developing a graphical interface for both input and output. Consequently, the committee recommended that an implementation study proposal be prepared with the goal of developing the desired graphical interface.

CHAPTER 1. INTRODUCTION AND LITERATURE SURVEY

The basis of a rational pavement design procedure is the analysis of a layered pavement system. This pavement analysis system should include models of the solid media, material models for the pavement and subgrades, kinematics conditions, and more importantly, realistic simulations of the loading conditions. Considerable efforts have been reported in recent years which use a linear elastic analysis to obtain the basic informations needed for the development of such a design procedure. In the analysis, the pavements are assumed to be linearly or bi-linearly elastic, homogeneous, and isotropic. Loading are usually taken to be static. As these highly simplified assumptions do not correspond to the real pavement properties, many cases exist where measured pavement responses in term of deflection or strain do not agree with those predicted by a linearly elastic model. Shift factors, for example, have to be introduced to compensate the differences. These factors may have a wide range in magnitude which make their reliability and accuracy susceptible to large errors.

Continued progress toward rational pavement design that would effectively simulate various types of static and dynamic loadings, and properties of conventional and emerging paving materials suggests that an efficient three dimensional finite element computer program would be highly desirable. In the past, such programs were restrictive and complicated, and required the use of large mainframe computer systems. The advent of new algorithms and hardwares in recent years indicate that such an analysis can be performed efficiently on a personal computer system. The primary motivation of this project is then to implement such an efficient computational analysis system, which may serve as the foundation of a rational pavement design procedure.

1.1 Explicit Approach of Finite Element Analysis

In this project, an explicit approach of finite element analysis is adopted as the basic methodology for the development of a pavement analysis system. The basic ingredients of this approach consist of a vector formulation of finite elements to obtain a set of equations of motion, an explicit time-integration procedure to find solutions of the equations of motion, and a co-rotational formulation to handle large rotations.

a. A Vector Formulation of Finite Elements:

In the traditional finite element analysis, the stiffness and force matrices are calculated based on structural discretization. These matrices are assembled to form a system of simultaneous linear equations for the solution of nodal displacements. For dynamic problems, a similar procedure can be used to find the mass matrix, and a system of differential equations of motion are obtained.

Instead of using the matrix formulation, a transient formulation developed earlier by Key, Belytschko and Hallquist was adopted. In the formulation, the continuous medium is approximated by discrete mass particles. Using the standard finite element analysis, energy equivalent internal and external forces are found. They are the forces applied on the mass particles. Newton's law of motion and time integrations complete the formulation to determine the acceleration, velocity and the displacement of each particle for a particular time increment.

Since the discretization considers only the lumped mass particles and lumped forces acting on the particles, the formulation is written in a vector form. One of the distinct

advantages of a vector formulation is that the code development is considerably simpler than a matrix analysis.

Note that the equations of motion for each mass particle are essentially solved independently. Mass particles are related to one another only through the internal forces. This property of a mass-force system offers considerable advantages:

* The motion of each particle is calculated independently. The equations of motion yield total absolute motion of the particle, including both the rigid body motion and deformable motion.
* The system is an assemblage of independent particles, regardless whether these particles form a unit body or multiple bodies. The only difference lies in the calculation of internal forces. Hence, the algorithm automatically permits multiple bodies and body separations.
* Since material properties are included in the computation of internal forces, inelastic and discontinuous material properties are easy to handle.
* Adding or subtracting a set of nodes do not affect the original discretization. Hence, creating new surfaces or eliminating portions of the medium due to failure presents no numerical problem. Hence, the algorithm can be modified to simulate construction process and fracture.
* Numbering of the nodes has no effect on the computation. Again this is convenient for creating or eliminating surfaces and bodies.
* Since the interactions between two components are through the internal forces, the co-existence of soft and stiff components do not present numerical problems. The extreme case is that some components have zero stiffness as a result of failure.
* It is easy to handle different types of structural component without the need for complicated assemblage process.
* Coding for the algorithm is simple. The resulting program is short and compact.

Since Newton's law of motion forms the basis, the algorithm is a transient procedure by nature. Static solutions can be obtained by attenuating the motion through the use of dynamic relaxation and apply external loads incrementally.

b. Explicit Time Integration:

To avoid the complexity of iterations, a simple explicit time integration formulation is suggested to find velocity and displacement for each time increment. This simplifies the implementation for complicated material models, changing constraint conditions and loading conditions. Inelastic and failure conditions become much simpler to incorporate. However, small time and force increments are required for numerical stability. This leads to longer computational times in general.

c. A Co-rotational for Large Rotations:

To develop a more accurate approach to handle large deformation and yet simple in computation, a co-rotational formulation of the kinematics is introduced. For each time increment, the motion is separated into two parts: a pure rigid body rotation of the entire finite element and a deformation of the element which characterizes the shape changes. A convected local coordinate system is introduced which is updated for each time increment. The strain tensors and hence the corresponding stress tensors are formulated in the local coordinates. For small changes in the element geometry, linear stress and strain relationships can be assumed despite the large overall changes in geometry due to rotations.

1.2 OBJECTIVES

The primary objective is to develop a three-dimensional finite element program for the analysis of general pavement problems. The program considers conventional static and dynamic loading conditions including harmonic excitations, pulse loadings, ramp loadings, and multiple step loadings. Provisions are made for the convenience of handling non-conventional loadings such as falling weights and general time-dependent load histories generated from non-destructive testings used for pavement structural evaluation. One particular important aspect of the program is to develop a general material library. Common soil and asphalt material models in the form of linear and nonlinear elastic materials, elastic-plastic materials with hardening, and viscoelastic materials are included in the material library.

The program should be coded such that further expansions of the load and material libraries only require minimum effort.

1.3 LITERATURE REVIEW

a. Algorithm for Pavement analysis and design using 3D finite elements:

A TRIS data base search was performed. Most of the current work can be categorized as the use of finite element for the study of special features of pavements, rather than the development of a suitable finite element algorithm for the pavement analysis. The majority of the work are also based on the use of existing commercial codes.

A study at University of Minnesota (Koubaa and Krauthammer 1990) was reported. The goal is to develop an analysis method combined with a non-destructive testing procedure for the evaluation of load transfer for joints in concrete pavements. The basic analysis involved a frequency response analysis by dynamically loading the joints. A three-dimensional finite element method was used to analyze various joint conditions for load transfer ranging from full to partial load transfer.

Stoner, et al. 1990 reported on research plans to develop a 3D finite element program to study concrete pavements and to simulate truck actions. The objectives were to develop a truck simulation model, to develop a model for doweled concrete pavement, and to implement these two models in an interactive fashion. The analysis was intended to obtain performance relationships based on damages predicted by the program. Actual damages recorded on Interstate 80 were used to verify the predictions.

Barksdale 1971 reported a study of compressive stress pulse at different depths in a flexible pavement. Loadings for variable vehicle speed were considered. A series of pseudo-dynamic linear and nonlinear elastic analyses were conducted. It was concluded that linear elastic finite element was adequate for the analysis. Dynamic effects including damping and inertial forces were neglected in the study and hence, a correction factor was introduced in order to match the results of AASHTO road tests.

Paterson 1983 reported on the finite element analysis of an asphalt overlay of a cracked airport concrete pavement in combination with a thin interlayer of elastomeric asphalt. Predictions were obtained in terms of an equivalent thickness of asphalt overlay which yields the same performance of the interlayer system.

Measured pavement deflections in combination with a three-dimensional finite element analysis were used to evaluate overlay requirement and pavement performance in a study reported by Bala and Kennedy 1986. The reponse predictions of the finite element analysis were calibrated against surface deflections measured by a deflectograph. The calibration process included adjustments determined through a parametric study. The properties assumed for the materials in the analysis were compared with laboratory determined dynamic test results for in situ samples.

Zaghloul and White 1993a reported results of a three-dimensional dynamic finite element analysis of flexible pavements. The analysis simulated actual truck loads moving at different speeds. Linear and non-linear material properties were used to model different paving materials and subgrades. An extended Drucker-Prager model was used to model granular materials, and an extended Cam-Clay model was used for the clayey soils. Asphalt mixtures were modeled as viscoelastic materials. Including these material models lead to the capability to obtain accurate elastic and plastic pavement responses. With this capability, they are able to predict or to interpret pavement performances under a variety of loading conditions and for different material characteristics.

The 3D finite element analysis was verified by comparing the predictions with a multi-layer elastic system, assuming linear elastic properties and static loads. A linear correlation was found between the results obtained by the finite element predictions and the multi-layer elastic analysis. To verify the dynamic, nonlinear finite element analysis, the results were compared with actual measurements of pavement deflection. Agreement at 95% confidence level was obtained between the deflection predictions and the measurements.

A sensitivity study was performed by using the 3D finite elements for the effect of cross section and load parameters on pavement responses. It was found that the speed of

moving vehicle load has a significant effect on elastic and plastic pavement responses. The confinement of shoulders has the effect of reducing pavement deflections. A crack along the pavement/shoulder joint results in an increase in pavement deflection. Temperature affects the asphalt layer and hence the overall pavement responses. The loading time and the rate of loading were found to have significant effect also. When a subgrade is subjected to a high stress level, higher than its yield stress, rutting increases significantly.

The effects of different load attributes, axle load and spacing, number of axles, and number of wheels, as well as cross section attributes, subgrade type, material properties and the type of deep foundation were also studied, and found to be significant to pavement responses.

In a separate study founded by INDOT, Zaghloul and White 1993b reported the study of heavy loads and their effects on the Indiana highway network. It was decided to evaluate the damages induced by the heavy loads by their LEF's. Available LEF's were found not suitable for the study. New LEF's were developed based on the total pavement deformation at the pavement surface. A 3D dynamic finite element model was used to perform the analysis. A comparison of the Purdue LEF's and the AASHTO LEF's was made and no significant difference was found. Purdue LEF's consider different load and cross section parameters, while the AASHTO LEF's do not. In addition, Purdue LEF's were developed based on analytical models, which can be extended to consider a wide range of other variables.

The study reported by Zaghloul and White have utilized a commercial general purpose finite element code. Due the general nature of the code, the run time required to use a commercial code for a particular problem is usually much longer than what is necessary.

More importantly, the flexibility in application is limited. Modifications to tailor the code for extended pavement applications are difficult.

b. Explicit Finite Element Algorithms:

Classical static solutions of finite element analysis generally adopt an implicit approach. The formulations are reduced to a set of simultaneous equations (or matrix equations) to find the displacement and stress values at discrete points. The details of an implicit approach are well documented in the textbooks, for example, (Bathe 1982). To extend the formulations to handle dynamic analysis due to impact loadings, or to perform nonlinear analysis due to inelastic behaviors of material and large changes in structural geometry, one encounters considerable difficulties in the development of a reliable solution algorithm. Existing algorithms generally involve complicated integration procedures as well as repeated iterations. As the result, the application of an implicit approach for general transient inelastic analysis is somewhat awkward. Code development also becomes difficult and inefficient (Bathe, et al. 1975).

To circumvent many of these difficulties, explicit approaches have been proposed and implemented. One which is widely used considers the structure as a set of discrete mass particles. By using the particle mass equations of motion including external and internal forces, the formulation reduces the finite element model to a set of vector equations. Thus, the approach avoids the assemblage and the storage of large matrix equations, and thus the solution algorithms are much simplified. The vector equations generally calculate particle accelerations due to external loadings. Displacements and corresponding stresses can be found by a time integration procedure.

The explicit approach has been proved to be particularly suitable for impact analysis for which a time series of the loading history is prescribed. It is also convenient to study highly nonlinear and discontinuous material properties, as well as very large displacements often associated with nonlinear properties (Key 1974, Oden and Key 1973). A large number of general purpose computer codes have been developed in recent years, mostly tailored for applications in the defense industry and for the safety analysis of nuclear power plants. For example, STRAW was developed by Argonne National Laboratory (Kennedy et al. 1985), and DYNA2D and DYNA3D by Lawrence Livermore Laboratory (Hallquist 1982). Commercial codes including ABAQUS have also issued explicit versions recently.

Fundamental studies of explicit approaches have been carried out at Purdue Civil Engineering during the past decade. Algorithms for large deflection of space frames (Saha and Ting, 1983), for concrete fracture analysis (Saha 1983), and for the failure analysis of concrete slabs, folded plates, and shells (Labbane 1991) have been reported. They include a variety of structural finite element formulations. Special algorithms have been studied for elastic-viscoplastic materials (Labbane and Ting 1991), for extremely large deflections (Rice and Ting 1991), and for material fragmentation process (Rice and Ting 1992). Computer codes using the explicit approach for all the studies have shown to be efficient and short. They generally require a small memory and hence, are suitable for personal computers. In general the explicit approaches are coded for transient dynamic analysis. A dynamic relaxation procedure is used to find static solutions.

CHAPTER 2 CO-ROTATIONAL APPROACH AND EQUATION OF MOTION

In this chapter, the traditional co-rotational approach for a plane element subjected to displacements is formulated. The co-rotational approach is used to treat large rotations of a deformable solid element.

Let the solid be modeled by elements. A set of rectangular coordinates is attached to each element, which translates and rotates with the element during the deformation process but does not deform. Thus, this set of convected or co-rotational coordinates is always an orthogonal system.

If the solid element is subjected to large displacements, we assume that the translation and rotation of the convected coordinates can be large. However, the deformations described in the convected coordinate system are small. This forms a large rotation-small deformation theory, where small strain theory and linear stress-strain relationships can be used.

Schematically, we may illustrate the co-rotational approach by considering three structural configurations, as shown in Figure 2.1. They are (a) the undeformed geometry $\boldsymbol{X}$ at time $t = 0$, (b) the deformed geometry $\boldsymbol{x}$ at time t, and (c) a convected geometry $\hat{\boldsymbol{x}}$ at time t. Hence,

$$\mathbf{dx} = \mathbf{F}\,\mathbf{dX}\,, \quad \mathbf{d\hat{x}} = \mathbf{T}\,\mathbf{dx}$$

$\boldsymbol{T}$ and $\boldsymbol{F}$ are transformation matrices. If $\boldsymbol{T}$ is a pure rotation,

$$d\mathbf{x} = \mathbf{T}^T d\hat{\mathbf{x}}$$

where $\mathbf{T}^T$ is the transpose matrix of T. And

$$d\hat{\mathbf{x}} = \mathbf{T}\,\mathbf{F}\,d\mathbf{X} = \hat{\mathbf{F}}\,d\mathbf{X}$$

The strain induced by the transformation $\hat{\mathbf{F}}$ can be treated by an infinitesimal theory.

Putting in the context of finite elements, we consider a nodal displacement vector $\mathbf{d}_e$ in global coordinates and a nodal displacement vector $\hat{\mathbf{d}}_e$ in convected coordinates. They are related by a rotation matrix T as

$$\hat{\mathbf{d}}_e = \mathbf{T}\,\mathbf{d}_e \tag{2.1}$$

We further assume that the total displacement vector can be written as the sum of a deformation vector $\mathbf{d}_e^d$ and a rigid body motion vector $\mathbf{d}_e^r$,

$$\mathbf{d}_e = \mathbf{d}_e^d + \mathbf{d}_e^r \tag{2.2}$$

The corresponding vector in convected coordinates are

$$\hat{\mathbf{d}}_e = \hat{\mathbf{d}}_e^d + \hat{\mathbf{d}}_e^r \tag{2.3}$$

and

$$\hat{\mathbf{d}}_e^d = \mathbf{T}\,\mathbf{d}_e^d \quad , \quad \hat{\mathbf{d}}_e^r = \mathbf{T}\,\mathbf{d}_e^r$$

For small deformation, the strain tensor depends on $\hat{\mathbf{d}}_e^d$ only.

2.1 Rotation Matrix for Plane Solid Elements

Consider a four-node plane element shown in Figure 2.2(a). $(\hat{x},\hat{y})$ is a set of local, or convected, coordinates with its origin at the center of the element. (x, y) is denoted as the global coordinates. (u, v) is the displacement vector in the global coordinates. From time t to time t', the change of the orientation of $(\hat{x},\hat{y})$ coordinates with respect to the rigid body rotation angle θ of the element can be described as shown in Figure 2.2(b) such as

$$\theta = \frac{1}{4}(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) \tag{2.4}$$

or

$$\theta = \frac{1}{4}\sum_{i=1}^{4}\alpha_i \tag{2.5}$$

where α_i is the rotational angle of the *ith* local nodal position vector from time t to time t'. By using cross product, α_i can be calculated as

$$\underset{\sim}{l} \times \underset{\sim}{l}' = \left|\underset{\sim}{l}\right| \cdot \left|\underset{\sim}{l}'\right| \cdot \sin\alpha_i \, \bar{k} \tag{2.6}$$

or

$$\alpha_i = \sin^{-1}\left\{\frac{1}{\left|\underset{\sim}{l}\right| \cdot \left|\underset{\sim}{l}'\right|}\left[(x_i - x_c)(y_i + v_i - y'_c) - (y_i - y_c)(x_i + u_i - x'_c)\right]\right\} \tag{2.7}$$

where

$$\left|\underset{\sim}{l}\right| = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}$$

$$\left|\underset{\sim}{l}'\right| = \sqrt{(x_i + u_i - x'_c)^2 + (y_i + v_i - y'_c)^2}$$

$$x_c = \frac{1}{4}(x_1 + x_2 + x_3 + x_4)$$

2.2 Kinematics for a Plane Solid

The displacement vector (u, v) can be decomposed to a rigid body component and a deformation component, i.e.,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u' \\ v' \end{bmatrix} + \begin{bmatrix} u^d \\ v^d \end{bmatrix} \tag{2.13}$$

Also, the rigid body vector is the sum of a translation and a rotation components,

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} u^t \\ v^t \end{bmatrix} + \begin{bmatrix} u^\theta \\ v^\theta \end{bmatrix} \tag{2.14}$$

as shown in Figure 2.4. Using the rotation matrix, $\boldsymbol{R}$, we get the deformation component in convected coordinates

$$\begin{bmatrix} \hat{u}^d \\ \hat{v}^d \end{bmatrix} = \mathbf{R} \begin{bmatrix} u^d \\ v^d \end{bmatrix} \tag{2.15}$$

assumed an infinitesimal strain theory.

Note that, for the element deformation, the rigid body motions should be separated from the total nodal displacements. Since $\hat{x}$-axis passes through the element center, the nodal displacement components $\hat{d}_{cx}$ and $\hat{d}_{cy}$ of the central point of the element are induced by rigid body motions. This deformation displacement vector $(\hat{u}^d, \hat{v}^d)$ is related to nodal deformation displacements by shape functions. In finite element technique, the isoparametric formulation then is introduced for this term.

Isoparametric formulation uses the same shape function to define the geometric shape of the element and to describe the displacements within the element. The function is formulated using a natural coordinate system, or so-called *s-t* domain,

which is a transformation mapping domain of the convected coordinates. Since there are only two nodes for each side of a four-node element, a linear shape function for the displacements and the nodal coordinates along the s or t element boundary is assumed. Multiplication of the linear functions in s and t yields the shape functions for the four-node element in the form

$$\begin{aligned} \hat{u} &= a_1 + a_2 s + a_3 t + a_4 st \\ \hat{v} &= a_5 + a_6 s + a_7 t + a_8 st \end{aligned} \tag{2.16}$$

$$\begin{aligned} \hat{x} &= a_1 + a_2 s + a_3 t + a_4 st \\ \hat{y} &= a_5 + a_6 s + a_7 t + a_8 st \end{aligned} \tag{2.17}$$

By solving for the eight a_i unknowns in terms of nodal coordinates, the following equations are established

$$\begin{Bmatrix} \hat{x} \\ \hat{y} \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \begin{Bmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{x}_2 \\ \hat{y}_2 \\ \hat{x}_3 \\ \hat{y}_3 \\ \hat{x}_4 \\ \hat{y}_4 \end{Bmatrix} \tag{2.18}$$

$$\begin{Bmatrix} \hat{u} \\ \hat{v} \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \begin{Bmatrix} \hat{u}_1 \\ \hat{v}_1 \\ \hat{u}_2 \\ \hat{v}_2 \\ \hat{u}_3 \\ \hat{v}_3 \\ \hat{u}_4 \\ \hat{v}_4 \end{Bmatrix} \tag{2.19}$$

A schematic diagram for the transformation mapping is shown in Figure 2.5, and the shape function, N_i , are now

$$N_1 = \frac{1}{4}(1-s)(1-t) \qquad N_2 = \frac{1}{4}(1+s)(1-t)$$

$$N_3 = \frac{1}{4}(1+s)(1+t) \qquad N_4 = \frac{1}{4}(1-s)(1+t) \tag{2.20}$$

which satisfies the displacement continuity at all boundaries and $N_1 + N_2 + N_3 + N_4 = 1$.

The element strains based on the convected geometry are given by

$$\begin{Bmatrix} \varepsilon_{\hat{x}} \\ \varepsilon_{\hat{y}} \\ \gamma_{\hat{x}\hat{y}} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial \hat{u}}{\partial \hat{x}} \\ \dfrac{\partial \hat{v}}{\partial \hat{y}} \\ \dfrac{\partial \hat{u}}{\partial \hat{y}} + \dfrac{\partial \hat{v}}{\partial \hat{x}} \end{Bmatrix} \tag{2.21}$$

or in s-t domain

$$\begin{Bmatrix} \varepsilon_{\hat{x}} \\ \varepsilon_{\hat{y}} \\ \gamma_{\hat{x}\hat{y}} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} \dfrac{\partial \hat{y}}{\partial t}\dfrac{\partial \hat{u}}{\partial s} - \dfrac{\partial \hat{y}}{\partial s}\dfrac{\partial \hat{u}}{\partial t} & 0 \\ 0 & \dfrac{\partial \hat{x}}{\partial s}\dfrac{\partial \hat{v}}{\partial t} - \dfrac{\partial \hat{x}}{\partial t}\dfrac{\partial \hat{v}}{\partial s} \\ \dfrac{\partial \hat{x}}{\partial s}\dfrac{\partial \hat{v}}{\partial t} - \dfrac{\partial \hat{x}}{\partial t}\dfrac{\partial \hat{v}}{\partial s} & \dfrac{\partial \hat{y}}{\partial t}\dfrac{\partial \hat{u}}{\partial s} - \dfrac{\partial \hat{y}}{\partial s}\dfrac{\partial \hat{u}}{\partial t} \end{bmatrix} \tag{2.22}$$

Substitute Equation (2.20) into Equation (2.22b), we get

$$\underset{\sim}{\varepsilon} = \underset{\sim}{B}\,\underset{\sim}{d} \tag{2.23}$$

where $\underset{\sim}{d}$ is given by Equation (2.10) and

$$\underset{\sim}{B} = \frac{1}{|J|}\begin{Bmatrix} \underset{\sim}{B}_1 & \underset{\sim}{B}_2 & \underset{\sim}{B}_3 & \underset{\sim}{B}_4 \end{Bmatrix} \tag{2.24}$$

$|J|$ is the Jacobian determinate defined as

$$|J| = \begin{vmatrix} \frac{\partial \hat{x}}{\partial s} & \frac{\partial \hat{y}}{\partial s} \\ \frac{\partial \hat{x}}{\partial t} & \frac{\partial \hat{y}}{\partial t} \end{vmatrix}$$

Or explicitly,

$$|J| = \frac{1}{8}\{\hat{x}_1 \quad \hat{x}_2 \quad \hat{x}_3 \quad \hat{x}_4\}\begin{bmatrix} 0 & 1-t & t-s & s-1 \\ t-1 & 0 & s+1 & -s-t \\ s-t & -s-1 & 0 & t+1 \\ 1-s & s+t & -t-1 & 0 \end{bmatrix}\begin{Bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{Bmatrix} \quad (2.25)$$

Also,
$$B_i = \begin{bmatrix} \frac{\partial \hat{y}}{\partial t}\frac{\partial N_i}{\partial s} - \frac{\partial \hat{y}}{\partial s}\frac{\partial N_i}{\partial t} & 0 \\ 0 & \frac{\partial \hat{x}}{\partial s}\frac{\partial N_i}{\partial t} - \frac{\partial \hat{x}}{\partial t}\frac{\partial N_i}{\partial s} \\ \frac{\partial \hat{x}}{\partial s}\frac{\partial N_i}{\partial t} - \frac{\partial \hat{x}}{\partial t}\frac{\partial N_i}{\partial s} & \frac{\partial \hat{y}}{\partial t}\frac{\partial N_i}{\partial s} - \frac{\partial \hat{y}}{\partial s}\frac{\partial N_i}{\partial t} \end{bmatrix} \quad (2.26)$$

and
$$\frac{\partial \hat{y}}{\partial t} = \frac{1}{4}\left[\hat{y}_1(s-1) + \hat{y}_2(-1-s) + \hat{y}_3(1+s) + \hat{y}_4(1-s)\right]$$

$$\frac{\partial \hat{y}}{\partial s} = \frac{1}{4}\left[\hat{y}_1(t-1) + \hat{y}_2(1-t) + \hat{y}_3(1+t) + \hat{y}_4(-1-t)\right]$$

$$\frac{\partial \hat{x}}{\partial s} = \frac{1}{4}\left[\hat{x}_1(t-1) + \hat{x}_2(1-t) + \hat{x}_3(1+t) + \hat{x}_4(-1-t)\right]$$

$$\frac{\partial \hat{x}}{\partial t} = \frac{1}{4}\left[\hat{x}_1(s-1) + \hat{x}_2(-1-s) + \hat{x}_3(1+s) + \hat{x}_4(1-s)\right]$$

$$\frac{\partial N_1}{\partial s} = \frac{1}{4}(t-1) \qquad \frac{\partial N_1}{\partial t} = \frac{1}{4}(s-1)$$

$$\frac{\partial N_2}{\partial s} = \frac{1}{4}(1-t) \qquad \frac{\partial N_2}{\partial t} = \frac{1}{4}(-1-s)$$

$$\frac{\partial N_3}{\partial s} = \frac{1}{4}(1+t) \qquad \frac{\partial N_3}{\partial t} = \frac{1}{4}(1+s)$$

$$\frac{\partial N_4}{\partial s} = \frac{1}{4}(-1-t) \qquad \frac{\partial N_4}{\partial t} = \frac{1}{4}(1-s)$$

2.3 Principle of Virtual Work

Similar to the traditional finite element formulation, the equilibrium of structural system is defined by the principle of virtual work.

$$\sum_e \delta U_e - \sum_e \delta W_e = 0 \tag{2.27}$$

where the summation is carried out for all the elements and δU_e and δW_e are the element internal and external virtual work, respectively.

The element internal virtual work in terms of the element stress and strain can be written in the form

$$\delta U_e = \int_{\hat{V}_e} \delta \hat{\underline{\varepsilon}}^T \hat{\underline{\sigma}} \, d\hat{V} \tag{2.28}$$

For infinitesimal deformation,

$$d\hat{V} \equiv dV_0 \quad \text{and} \quad \hat{V}_e \equiv V_0$$

where V_0 is the undeformed volume. Using

$$\delta \hat{\underline{\varepsilon}}^T = (\delta \hat{\underline{d}}_e)^T \underline{B}^T$$

Equation (2.28) becomes

$$\delta U_e = (\delta \hat{\underline{d}}_e)^T \int_{V_0} \underline{B}^T \hat{\underline{\sigma}} \, dV_0 \tag{2.29}$$

Since

$$\delta U_e = (\delta \hat{\underline{d}}_e)^T \hat{\underline{f}}_e^{\text{int}} \tag{2.30}$$

where $\hat{\underline{f}}_e^{\text{int}}$ is the internal nodal forces associated with the nodal displacements.

Hence,

$$\hat{f}_e^{int} = \int_{V_0} B^T \hat{\sigma}\, dV_0 \tag{2.31}$$

Equation (2.31) can be written in terms of the material stress-strain relationships which will be described in Chapter four. We consider

$$\hat{\sigma} = C\hat{\varepsilon} \tag{2.32}$$

where C is the material property tensor. Then

$$\hat{f}_e^{int} = (\int_{V_0} B^T C B\, dV_0)\hat{d}_e \tag{2.33}$$

For a two dimensional solid with thickness t, we have

$$dV_0 = t\, dA_0 = t\, dx\, dy$$

Equation (2.33) becomes

$$\hat{f}_e^{int} = (t\iint B^T C B\, dx\, dy)\hat{d}_e$$

or in the s-t domain,

$$\hat{f}_e^{int} = (t\int_{-1}^{+1}\int_{-1}^{+1} B^T C B|J|\, ds\, dt)\hat{d}_e \tag{2.34}$$

For a four-node plane element, the integration of Equation (2.34) yields eight force components:

$$\hat{f}_e^{int} = \left\{\hat{f}_{1x} \quad \hat{f}_{1y} \quad \hat{f}_{2x} \quad \hat{f}_{2y} \quad \hat{f}_{3x} \quad \hat{f}_{3y} \quad \hat{f}_{4x} \quad \hat{f}_{4y}\right\} \tag{2.35}$$

Using the global nodal displacements d_e through the transformation, $\hat{d}_e = T d_e$, yields

$$\delta U_e = \delta d_e^T T^T \hat{f}_e^{int} = \delta d_e^T f_e^{int} \tag{2.36}$$

where

$$f_e^{\text{int}} = T^T \hat{f}_e^{\text{int}} \tag{2.37}$$

is the internal nodal force vector represented in the global coordinates.

The external virtual work may be treated as the sum of the work due to applied forces and the work due to inertia forces. For the inertia forces, if the element mass is lumped as concentrated masses located at the nodes of the element, then the corresponding virtual work due to inertia force is

$$\delta W_e^A = -\delta \hat{d}_e^T \hat{M}_e \ddot{\hat{d}}_e = -\delta d_e^T M_e \ddot{d}_e \tag{2.38}$$

where $\hat{M}_e$ is the lump mass matrix which is an invariant and diagonal. For a four-node plane element,

$$\hat{M}_e = \frac{1}{4}\rho A t \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.39}$$

where ρ is the mass density, A and t are the area and thickness, respectively, of the element.

The virtual work due to applied forces can be formulated following the traditional finite element approaches. Briefly, the forces given in global coordinates should be transformed to components in convected coordinates. The corresponding

nodal forces are evaluated according to the shape functions along the element boundary. After sub-assembling the force components to obtain the external force vector in convected coordinates, it is transformed back to global coordinates and we get

$$\delta W_e^B = \delta \underline{d}_e^T \underline{f}_e^{ext} \tag{2.40}$$

where $\underline{f}_e^{ext}$ is the external nodal force vector in global coordinates for the element. To consider the body force, since the element mass is lumped in the nodes, it is convenient to treat the body force as a concentrated external nodal force with respect to the gravity and to be included in Equation (2.40) as part of the $\underline{f}_e^{ext}$.

The total external virtual work for the element is then

$$\delta W_e = \delta W_e^A + \delta W_e^B = \delta \underline{d}_e^T (\underline{f}^{ext} - \underline{M}_e \underline{\ddot{d}}_e) \tag{2.41}$$

Using the principle of virtual work, we get

$$\sum_e \delta U_e - \sum_e \delta W_e = 0$$

or

$$\sum_e \delta \underline{d}_e^T \underline{f}_e^{int} - \sum_e \delta \underline{d}_e^T (\underline{f}_e^{ext} - \underline{M}_e \underline{\ddot{d}}_e) = 0 \tag{2.42}$$

Introducing a global (or assembled) nodal displacement matrix $\underline{d}$, Equation (2.42) becomes

$$\delta \underline{d}^T (\underline{F}^{int} - \underline{F}^{ext} + \underline{M} \underline{\ddot{d}}) = 0 \tag{2.43}$$

where $\underset{\sim}{F}^{int}$, $\underset{\sim}{F}^{ext}$ and $\underset{\sim}{M}$ are assembled internal nodal force matrix, external force matrix, and mass matrix, respectively. Or

$$\underset{\sim}{F}^{int} = \sum_e \underset{\sim}{f}_e^{int}$$

$$\underset{\sim}{F}^{ext} = \sum_e \underset{\sim}{f}_e^{ext} \tag{2.44}$$

$$\underset{\sim}{M} = \sum_e \underset{\sim}{M}_e$$

The square matrix $\underset{\sim}{M}$ remains to be diagonal after assemblage. As each of $\delta \underset{\sim}{d}$ is arbitrary, we get

$$F_\alpha^{int} - F_\alpha^{ext} + M_\alpha \ddot{d}_\alpha = 0 \quad , \quad \alpha = 1,2,3, \cdots\cdots n$$

where n is the total number of unknown nodal displacements. Or, the equations of motion are

$$\ddot{d}_\alpha = \frac{1}{M_\alpha} (F_\alpha^{ext} - F_\alpha^{int}) \tag{2.45}$$

Note that due to the use of lumped masses, the element formulation only requires the assemblage of force vectors. Furthermore, the equation of motion for each unknown can be calculated individually without the need for solving simultaneous equation. Thus, the current formulation combined with an explicit time integration to solve the equations of motion presents significant advantages in code development, data storage, and overall efficiency.

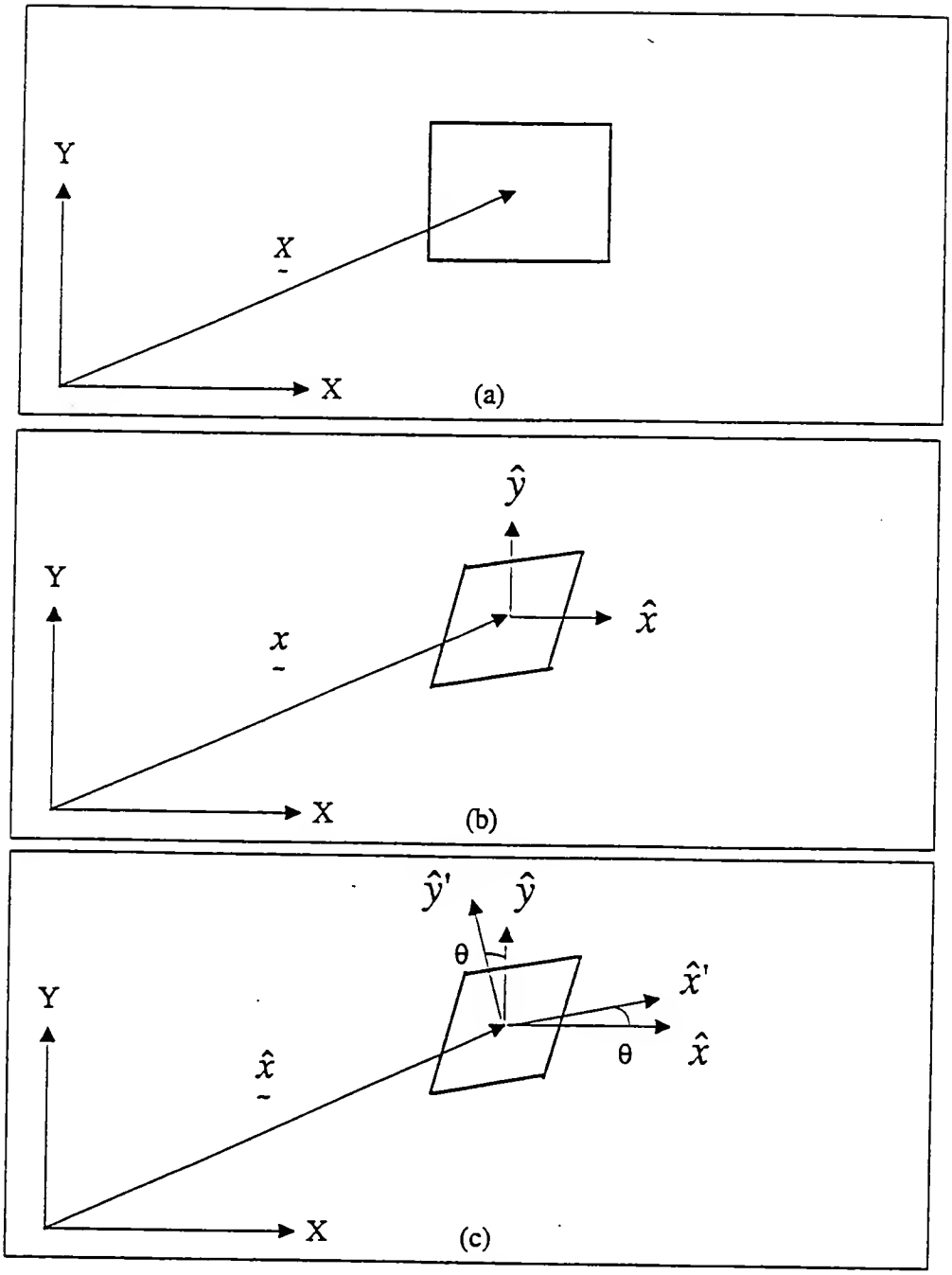

Figure 2.1 Three Structural Configurations in Co-rotational Approach (a) the Undeformed Geometry at time t = 0, (b) the Deformed Geometry at time t, (c) a Convected Geometry at time t.

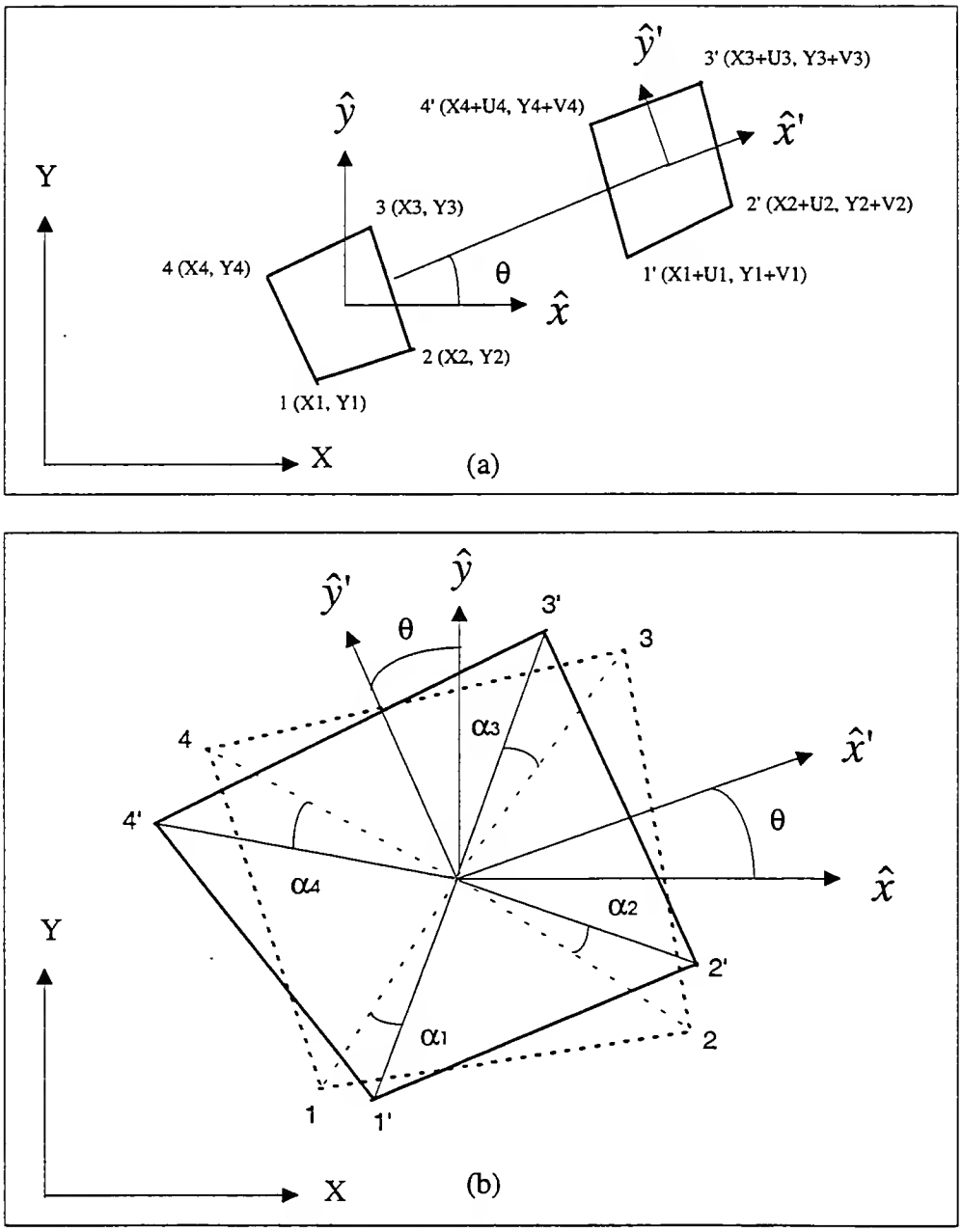

Figure 2.2 (a) Displacement and Rotation of Lines of a Plane Element in the X-Y Plane. (b) Pure Rotation.

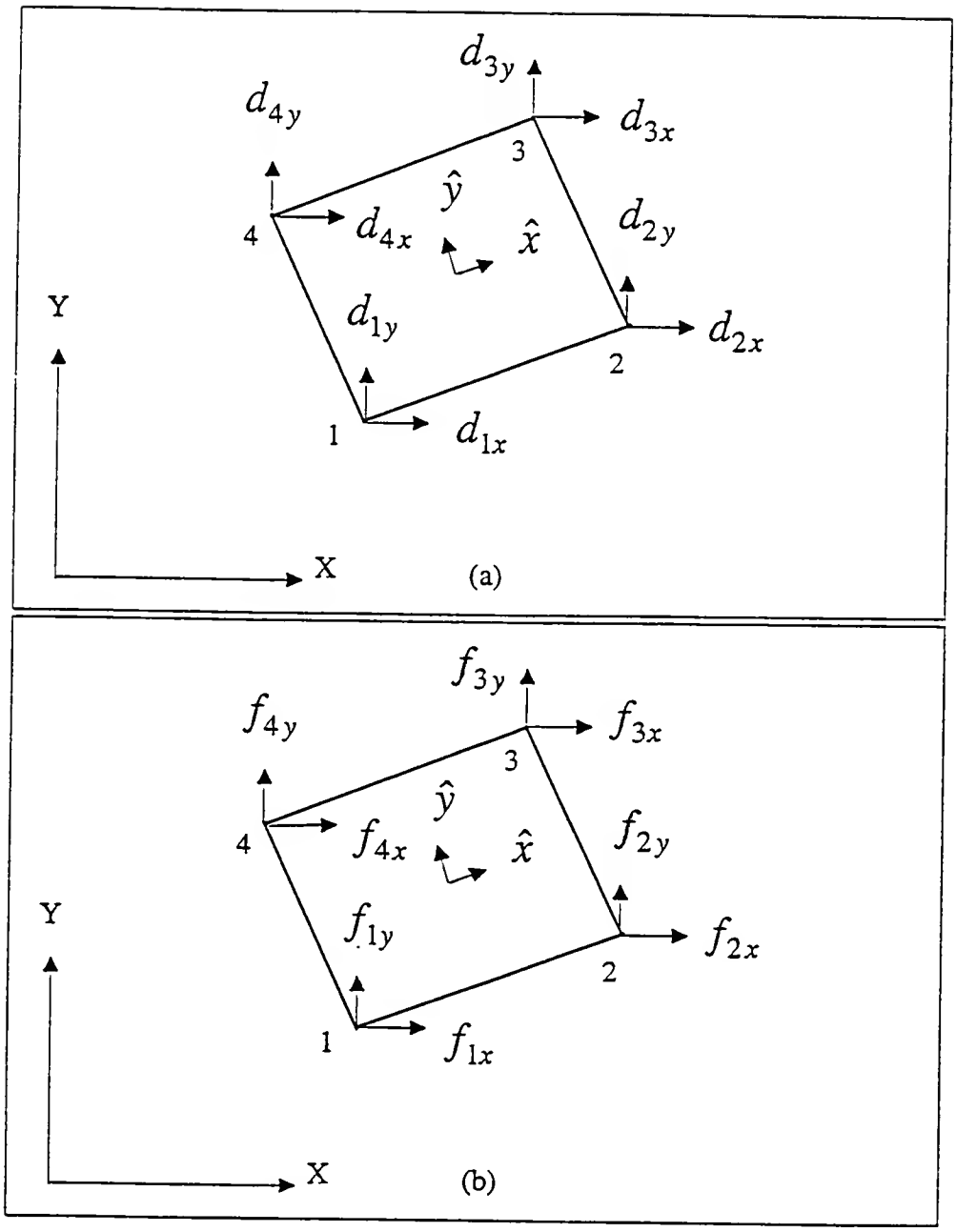

Figure 2.3 Degrees of Freedom of Nodes of a Four-Node Plane Element in X-Y Plane : (a) Displacement Part, (b) Force Part.

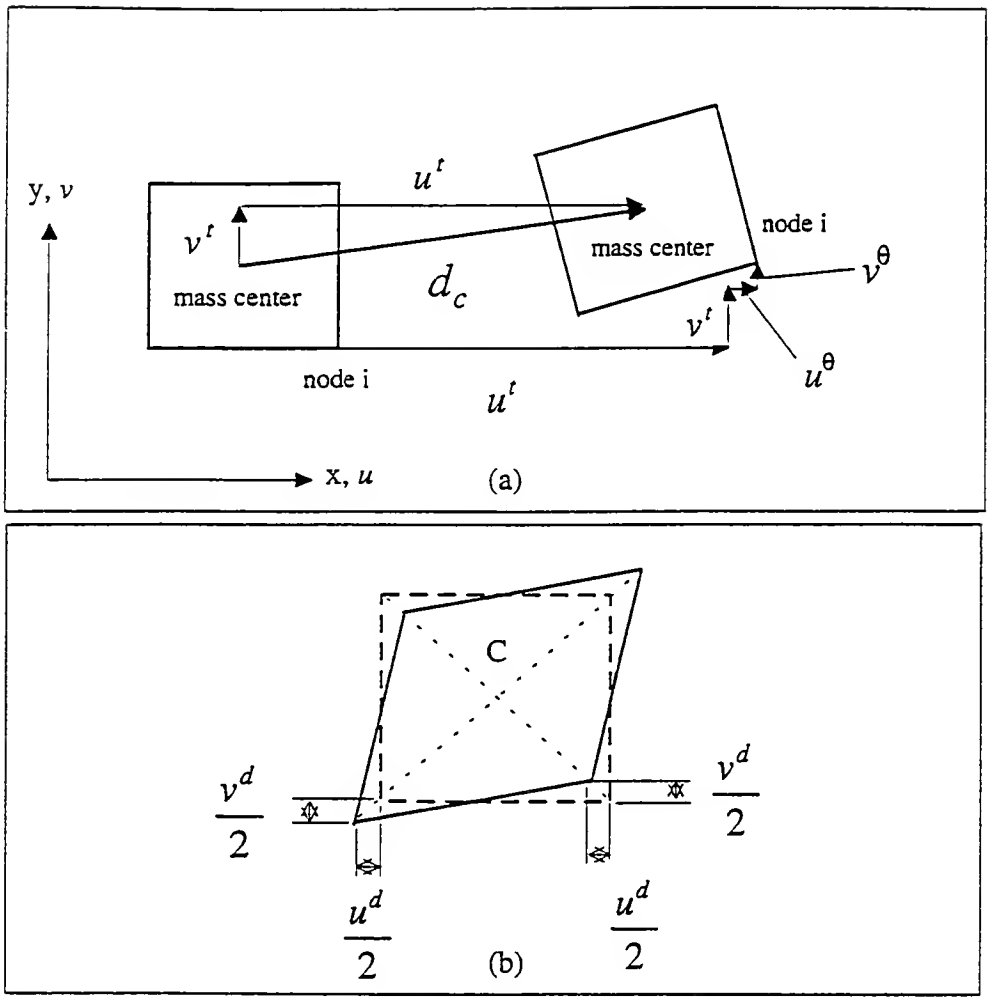

Figure 2.4 Decomposition of Displacements (a) Translation and Rotation without Strain, (b) Pure Strain without Translation and Rotation.

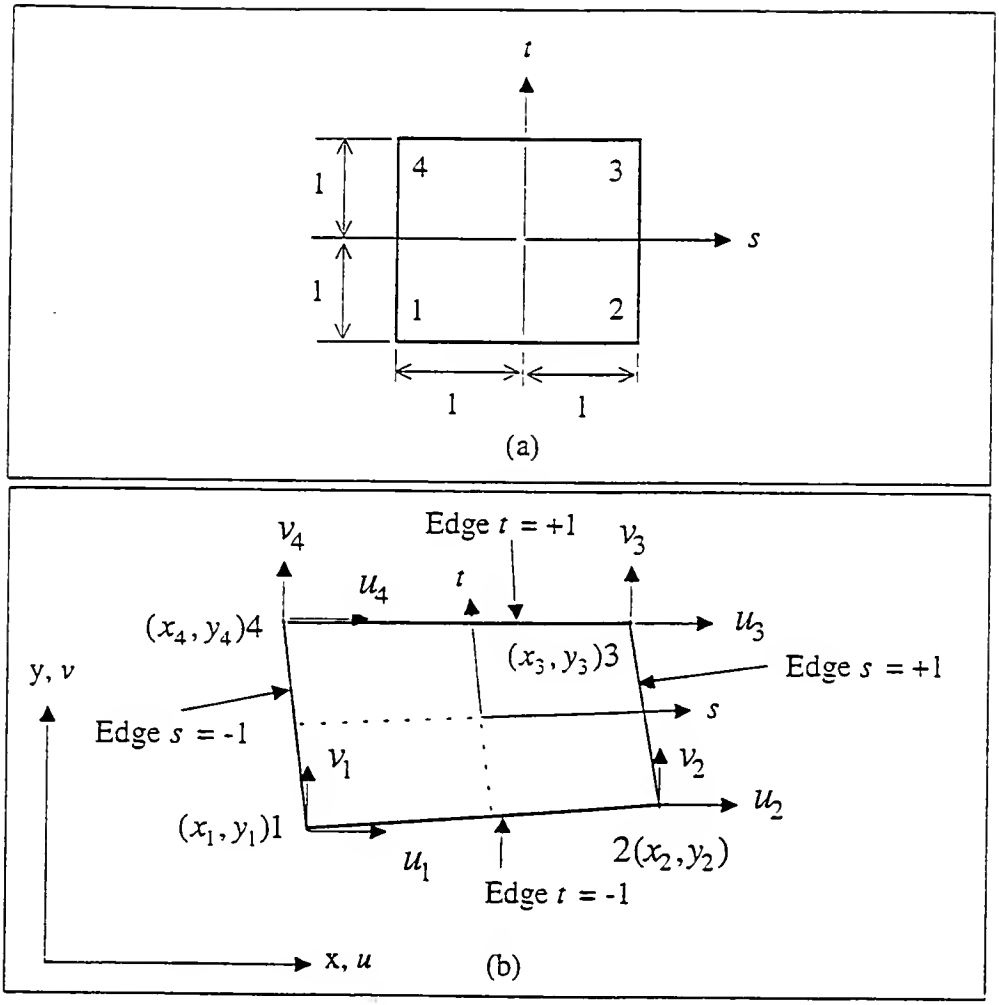

Figure 2.5 (a) Square Element in *s-t* Coordinates, (b) Quadrilateral Element in X-Y Coordinates are Mapped into a Square Element in *s-t* Coordinates.

CHAPTER 3 EXPLICIT TIME INTEGRATION

Direct integration procedures commonly used in solving structural dynamic problems may be categorized as explicit or implicit methods. The main difference is that explicit methods, such as central difference method, calculate the displacements at time $t + \Delta t$ based on the equation of motion at time t ; while the implicit methods, such as Houbolt, Wilson-θ and Newmark-β method [Bathe, 1982], use the equation of motion at time $t + \Delta t$. Hence, iterations are generally required for implicit procedures.

To choose a suitable method, one should consider the combined effect of discretizations on time and space, for example, calculating the natural frequencies of a structure. In the spatial discretization, if the mass matrix is obtained by a lumped approach, the frequencies are often underestimated. If ,however, the mass matrix is found by assuming a consistent approach, the frequencies are often overestimated. Similarly, in the time discretization, an explicit integration causes the frequencies to be overestimated; while the implicit integration underestimates the frequencies [Key, 1978]. Therefore, it seems that a favorable combination is to choose an explicit time integration method with lumped masses, or an implicit method with consistent masses.

An advantage of using an explicit method with lumped matrices is that there is no need to assemble stiffness matrices and does not need to solve simultaneous equations.

The solutions can be carried out in vector form which requires very small storage space. The drawback is that the procedure is conditionally stable. Hence, a small time increment is required for the calculation.

For a multi-degree-freedom system, the time integration technique used in this work is a second order central difference formulation which has the following relationships for accelerations and velocities :

$$\ddot{\mathbf{d}}_n = \frac{1}{\Delta t^2}\left(\mathbf{d}_{n+1} - 2\mathbf{d}_n + \mathbf{d}_{n-1}\right) \tag{3.1}$$

$$\dot{\mathbf{d}}_n = \frac{1}{2\Delta t}\left(\mathbf{d}_{n+1} - \mathbf{d}_{n-1}\right) \tag{3.2}$$

where Δt is the time increment, and the time span $t = n\,\Delta t$.

The equation of motion at the n - th time step has the form

$$\ddot{\mathbf{d}}_n = \mathbf{M}^{-1}\left(\mathbf{F}_n^{ext} - \mathbf{F}_n^{int}\right) \tag{3.3a}$$

If we wish to find a quasi-static solution through a dynamic relaxation procedure, a damping force may be added,

$$\ddot{\mathbf{d}}_n = \mathbf{M}^{-1}\left(\mathbf{F}_n^{ext} - \mathbf{F}_n^{int} - \mathbf{F}_n^{dmp}\right) \tag{3.3b}$$

The damping force may be written by assuming a standard Rayleigh damping [James, 1994] .

$$\mathbf{F}_n^{dmp} = \mathbf{C}\,\dot{\mathbf{d}}_n = \left(\alpha\,\mathbf{M} + \beta\,\mathbf{K}\right)\dot{\mathbf{d}}_n$$

in which α and β are constants, K is the stiffness matrix. In our calculation, since the global stiffness matrix is not available in the formulation, β is assumed to be zero. Or,

$$\mathbf{F}_n^{dmp} = \mathbf{C}\,\dot{\mathbf{d}}_n = \alpha\,\mathbf{M}\,\dot{\mathbf{d}}_n \tag{3.4}$$

Substituting Equation (3.1), (3.2) and (3.4) into (3.3b) yields

$$d_{n+1} = (\frac{2}{2+\alpha\Delta t})\frac{\Delta t^2}{M}(F_n^{ext} - F_n^{int}) + (\frac{4}{2+\alpha\Delta t})d_n - (\frac{2-\alpha\Delta t}{2+\alpha\Delta t})d_{n-1} \tag{3.5}$$

Note that using Equation (3.5), the displacements at $t + \Delta t$ are calculated by using the mass values and the external and internal forces of the previous time step. An important simplification can be introduced by assuming a diagonal mass matrix, since the inverse matrix can be obtained by the reciprocals of the diagonal mass values. All the calculations in Equation (3.5) involve vector operations only.

To start the solution process to calculate $\mathbf{d}_1$, we need $\mathbf{d}_{-1}$. Using the initial condition $\dot{\mathbf{d}}_0$, we may write

$$\dot{\mathbf{d}}_0 = \frac{1}{2\Delta t}(\mathbf{d}_1 - \mathbf{d}_{-1})$$

Or,

$$\mathbf{d}_{-1} = \mathbf{d}_1 - 2\Delta t\,\dot{\mathbf{d}}_0 \tag{3.6}$$

Substituting Equation (3.6) into Equation (3.5), $\mathbf{d}_1$ can be solved for.

Alternatively, we may use Equation (3.2) which gives

$$\mathbf{d}_1 = 2\Delta t\,\dot{\mathbf{d}}_0 + \mathbf{d}_{-1} \tag{3.7}$$

Substituting Equation (3.7) into Equation (3.1) for $n = 0$ gives

$$\ddot{\mathbf{d}}_0 = \frac{1}{\Delta t^2}(\mathbf{d}_1 - 2\mathbf{d}_0 + \mathbf{d}_{-1})$$

Or,

22

$$\mathbf{d}_{-1} = \mathbf{d}_0 - \Delta t\, \dot{\mathbf{d}}_0 + \frac{1}{2}\Delta t^2\, \ddot{\mathbf{d}}_0 \tag{3.8}$$

where $\ddot{\mathbf{d}}_0$ can be found from Equation (3.3b), such as

$$\ddot{\mathbf{d}}_0 = \mathbf{M}^{-1} (\mathbf{F}_0^{ext} - \mathbf{F}_0^{int} - \alpha \mathbf{M} \dot{\mathbf{d}}_0) \tag{3.9}$$

Since the time increment is small relative to the time span of interest, there is no significant difference in these two starting procedures.

In general, the explicit time integration is conditionally stable. That is, the time increment has to be smaller than a limit to avoid calculations becoming divergent. For a multi-degrees of freedom system, such a limit is difficult to obtain. Hence , an approximate limit on Δt is suggested [Hughes, 1979]

$$\Delta t < \frac{2}{\omega^{max}} \tag{3.10}$$

with ω^{max} being the highest frequency value for the elements in a structure.

CHAPTER 4 MATERIAL MODELS AND STRESS-STRAIN RELATIONSHIPS

In this Chapter, the basic material models which have been implemented and verified in the current computer codes are described. The stress-strain relationships are briefly reviewed. They include:

a. Linear Elastic Models:

Stress-strain relationships are given in the form of Hooke's Law. The elastic constants are Young's modulus and Poisson's ratio.

A generalization is to input tangent modulus to replace the Young's modulus. Effectively, this is a bi-linear elastic model or a nonlinear elastic model.

b. Plastic Models:

Stress-strain relationships are given for the Drucker-Prager yield criterion with associated flow rules. Three hardening rules are implemented: isotropic, kinematical, and mixed hardening rules. The special case of selecting a Mises yield criterion for the yield criterion and assuming an associated flow rule is included. Another alternative which is often used to model soil behaviors is to select a Drucker-Prager yielding function and assuming a non-associated flow rule with Mises function as the plastic potential. This combination has also been implemented.

Material constants inputs for soils are specified. They are the cohesion and the internal frictional angle.

c. Viscoelastic Models:

Stress-strain relationships are given for the viscoelastic material of Maxwell type. Implementation for other types of viscoelastic model such as the standard linear solid and Burger type are trivial. A general creep model based on a curve-fitting of the creep data can be included by a simple modification of the viscoelastic material subroutine.

4.1 Linear Elastic Stress-Strain Relations

A material possesses a linearly elastic property if it is elastic and the one-to-one relationship between the stresses and strains is linear. The generalized Hooke's law expressing the stress components as linear homogeneous functions of the strain components is given as

$$\sigma_{ij} = C_{ijkl}\,\varepsilon_{kl} \tag{4.1}$$

where C_{ijkl} is the fourth order material property tensor and can be expressed in terms of two independent material properties if material is isotropic, which means properties are not changed by orthogonal transformations of coordinates.

For isotropic case, the material property tensor can be expressed as

$$C_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) \tag{4.2}$$

where δ_{ij} is the Kronecker delta, λ and μ are Lamé constants in terms of Young's modulus E and Poisson's ratio ν or shear modulus G as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \qquad \mu = G = \frac{E}{2(1+\nu)} \tag{4.3}$$

Plane stress case assumes $\sigma_{3j} = 0$, whereas, plane strain case assumes $\varepsilon_{3j} = 0$. In geotechnical field, plane strain condition is more common.

4.2 Plastic Stress-Strain Relations

Plasticity theory extends elasticity theory when the state of stress satisfies the

yield criterion. It offers a mathematical description of the mechanical behavior of material in the plastic range. Since soils behave dependent on volumetric stress, the pressure-dependent plastic model, including Mohr-Coulomb and Drucker-Pracker yield criteria, are more popular for granular soils. In this research, the Drucker-Prager criterion is chosen. It is mentioned that in this section the sign convention for stresses is defined as "positive means compression" following the soil mechanics convention.

4.2.1 Drucker-Prager Yield Criterion

The Drucker-Prager yield function is given as the follows

$$f(I_1, J_2, \alpha, \kappa) = J_2 - (\alpha I_1 + \kappa)^2 = 0 \tag{4.4}$$

where I_1 = the first invariant of stress tensor $= \sigma_{ii}$,

J_2 = the second invariant of stress deviator tensor $= \frac{1}{2}\sigma_{ij}^{D}\sigma_{ij}^{D}$, in which

$$\sigma_{ij}^{D} = \sigma_{ij} - \frac{1}{3} I_1 \delta_{ij}, \text{ and}$$

α, κ = material constants.

In terms of the more familiar soil parameters, cohesion c and friction angle ϕ, the material constants α and κ can be taken either one set of the follows

$$\alpha = \frac{2\sin\phi}{\sqrt{3}(3+\sin\phi)} \text{ and } \kappa = \frac{6c\cdot\cos\phi}{\sqrt{3}(3+\sin\phi)} \tag{4.5}$$

or, $$\alpha = \frac{2\sin\phi}{\sqrt{3}(3-\sin\phi)} \text{ and } \kappa = \frac{6c\cdot\cos\phi}{\sqrt{3}(3-\sin\phi)} \tag{4.6}$$

Figure 4.1 shows the yield surface in the Haigh-Westergaard stress space and on the Pi-plane, respectively.

4.2.2 Incremental Stress-Strain Relationships

In plastic region, Equation (4.1) is no longer valid and is revised as

$$\sigma_{ij} = C^{ep}_{ijkl}\, \varepsilon_{kl} \tag{4.7}$$

where C^{ep}_{ijkl} is the elastic-plastic material property tensor and can be calculated from the elastic material property tensor, C_{ijkl}, and the plastic material property tensor, C^{p}_{ijkl}, by

$$C^{ep}_{ijkl} = C_{ijkl} - C^{p}_{ijkl} \tag{4.8}$$

Note that the plastic stress tensor is function of the current stresses, σ_{ij}, independent of the increment of stresses, $d\sigma_{ij}$. To find C^{p}_{ijkl}, the basic ingredients of the theory of plasticity must be carried out based on the given σ_{ij}, C_{ijkl}, $d\varepsilon_{kl}$, and f.

(1) Unit normal:
$$n_{ij} = \frac{\partial f / \partial \sigma_{ij}}{\left\| \partial f / \partial \sigma_{ij} \right\|} \tag{4.9}$$

where $\left\| \partial f / \partial \sigma_{ij} \right\| = \left(\frac{\partial f}{\partial \sigma_{ij}} \frac{\partial f}{\partial \sigma_{ij}} \right)^{1/2}$ = the magnitude of $\partial f / \partial \sigma_{ij}$. If stresses undergo plastic, two conditions must be satisfied: (a) $f = 0$ and (b) $d\sigma^{tr}_{ij}\, n_{ij} > 0$,

where $d\sigma_{ij}^{tr} = C_{ijkl}\, d\varepsilon_{kl}$.

(2) Strain decomposition: $$d\varepsilon_{ij} = d\varepsilon_{ij}^{e} + d\varepsilon_{ij}^{p} \tag{4.10}$$

in which the incremental small strain, $d\varepsilon_{ij}$, is decomposed into an elastic part, $d\varepsilon_{ij}^{e}$, and a plastic part, $d\varepsilon_{ij}^{p}$.

(3) Constitutive equation: $$d\sigma_{ij} = C_{ijkl}\, d\varepsilon_{kl}^{e} \tag{4.11}$$

(4) Flow rule: $$d\varepsilon_{ij}^{p} = \lambda \frac{\partial q}{\partial \sigma_{ij}} = \Lambda \frac{\partial q / \partial \sigma_{ij}}{\left\| \partial q / \partial \sigma_{ij} \right\|} \tag{4.12}$$

where Λ is the consistency parameter which is a scalar, and q is the plastic potential function. If $q = f$, this is so-called associated flow rule condition; if $q \neq f$, non-associated flow rule.

(5) Consistency condition: $$f = 0 \text{ and } df = 0. \tag{4.13}$$

A schematic consistency condition is shown in Figure 4.2.

Note that the use of the associated flow rule with Drucker-Prager model often overestimates the plastic dilation of the soil (shown in Figure 4.3); therefore, it is thus common to use a non-associated flow rule with Drucker-Prager to correct the problem. The von Mises yield function is then given as the plastic potential while Drucker-Prager yield function is for the yield surface. The von Mises yield function can be generated from the Drucker-Prager yield function by setting $c \neq 0$ and $\phi = 0$.

4.2.2.1 Linear Elastic-Perfectly Plastic Model

If the stress strain relationship of the material is modeled as an elastic-perfectly plastic behavior, shown in Figure 4.4, the elastic-plastic material property tensor for Drucker-Prager yield function with non-associated flow rule, as described previously, can be derived based on Equation (4.9) to (4.13) and concludes the forms

Drucker-Prager function: $$f(I_1, J_2, \alpha, \kappa) = J_2 - (\alpha I_1 + \kappa)^2 = 0 \tag{4.14}$$

von Mises function: $$q(J_2, \kappa) = J_2 - \kappa^2 = 0 \tag{4.15}$$

$$\Lambda = \frac{\underset{\sim}{n} : \underset{\sim}{C} : d\underset{\sim}{\varepsilon}}{\underset{\sim}{n} : \underset{\sim}{C} : \underset{\sim}{m}} \tag{4.16}$$

$$\underset{\sim}{m} = \frac{\partial q / \partial \underset{\sim}{\sigma}}{\left\| \partial q / \partial \underset{\sim}{\sigma} \right\|} = \frac{\underset{\sim}{\sigma}^D}{\sqrt{2}\kappa} \tag{4.17}$$

$$\underset{\sim}{n} = \frac{\partial f / \partial \underset{\sim}{\sigma}}{\left\| \partial f / \partial \underset{\sim}{\sigma} \right\|} = \frac{\underset{\sim}{\sigma}^D - 2\alpha(\alpha I_1 + \kappa)\,\underset{\sim}{1}}{\left\| \underset{\sim}{\sigma}^D - 2\alpha(\alpha I_1 + \kappa)\,\underset{\sim}{1} \right\|} \tag{4.18}$$

$$d\underset{\sim}{\sigma} = \underset{\sim}{C}^{ep} : d\underset{\sim}{\varepsilon} = (\underset{\sim}{C} - \underset{\sim}{C}^{p}) : d\underset{\sim}{\varepsilon} \tag{4.19}$$

$$\underset{\sim}{C}^{p} = \frac{\underset{\sim}{C} : \underset{\sim}{m} \otimes \underset{\sim}{n} : \underset{\sim}{C}}{\underset{\sim}{n} : \underset{\sim}{C} : \underset{\sim}{m}} \tag{4.20}$$

where $\underset{\sim}{1}$ is the second order unit tensor with components $1_{ij} = \delta_{ij}$.

4.3 Viscoelastic Stress-Strain Relations

Viscoelastic models characterize material behaviors which are time-dependent and temperature-dependent. Uniaxial tests or simple shear tests are commonly used to find the material constants. Hence, viscoelastic models are generally determined for uniaxial (Young's modulus) or shear (shear modulus) stress-strain relationships.

Extending the models for bi-axial or tri-axial stress conditions, assumptions have to be made. Two cases which are commonly assumed for application are taking the material to be incompressible and assuming the Poisson's ratio to be a constant.

In the program, we have taken Poisson's ratio to remain constant and the Young's modulus is replaced by a viscoelastic model.

4.3.1 Viscoelastic Models

Material characterized by two constants are known to be a Maxwell type, the relaxation function (Young's modulus) has the form

$$E = E_o \, e^{-\frac{t}{\tau}} \tag{4.21}$$

where τ is a relaxation time.

If four material constants are used, it is known as a Burger's model, where

$$E = E_1 \, e^{-\frac{t}{\tau_1}} + E_2 \, e^{-\frac{t}{\tau_2}} \tag{4.22}$$

4.3.2 Creep Models

Using the creep test data, time dependent behavior can be characterized by creep models

formulated in incremental form. Let incremental creep strain $\Delta\varepsilon_c$ be

$$\begin{aligned} \Delta\varepsilon_c &= \left(\frac{\sigma}{k}\right)^n \\ &= k\sigma^m t^n \end{aligned} \tag{4.23}$$

where n, m, k are constants obtained from curve-fitting.

Then, the incrementation stress $\Delta\sigma$ is

$$\begin{aligned} \Delta\sigma &= E(\Delta\varepsilon - \Delta\varepsilon_c) \\ &= E(1-\frac{\Delta\varepsilon_c}{\Delta\varepsilon})\Delta\varepsilon \end{aligned} \tag{4.24}$$

Implementation of the creep models is trivial, merely modify the tangent modulus E for each time or load increment.

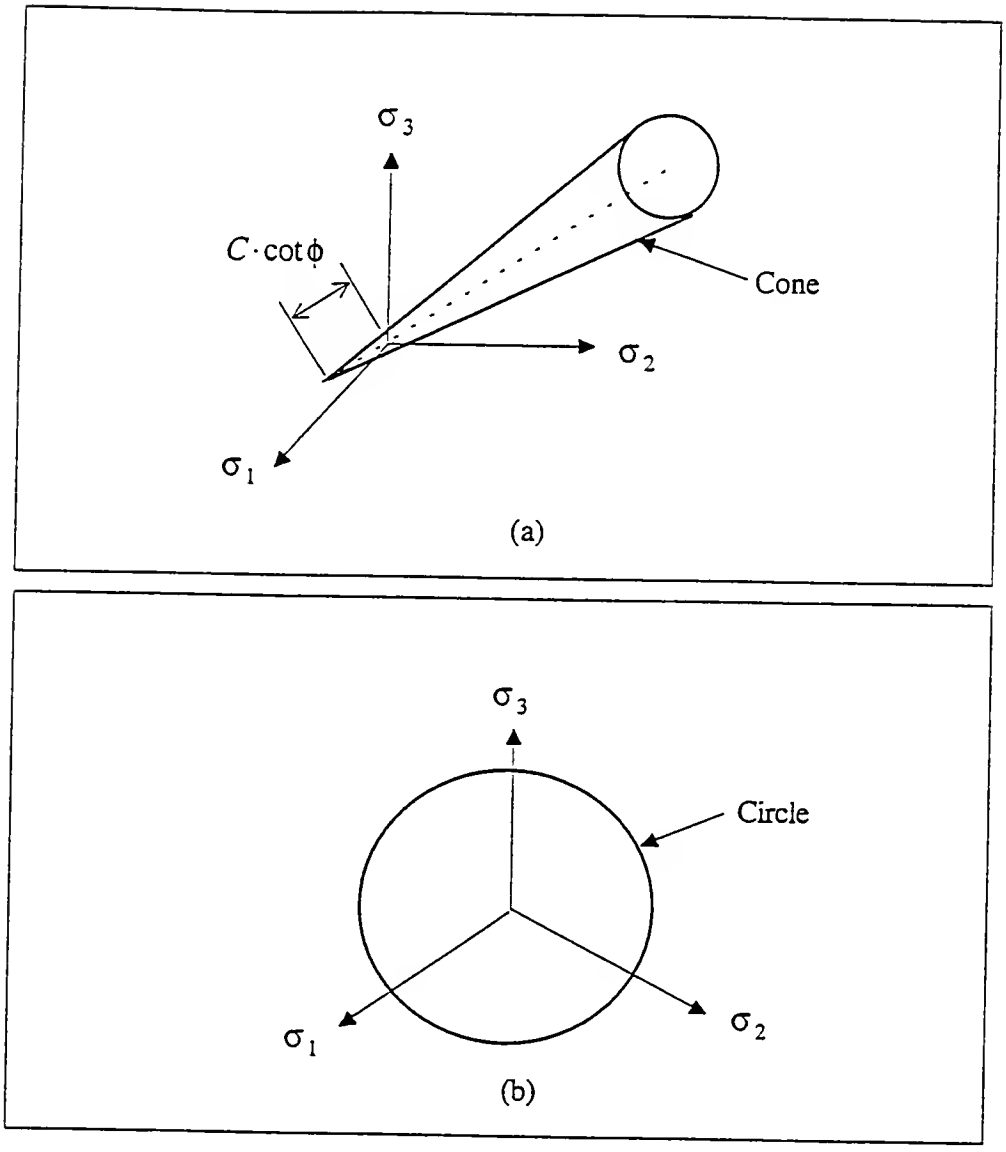

Figure 4.1 Drucker-Prager Yield Surface (a) in Haigh-Westergaard Stress Space, (b) on π Plane.

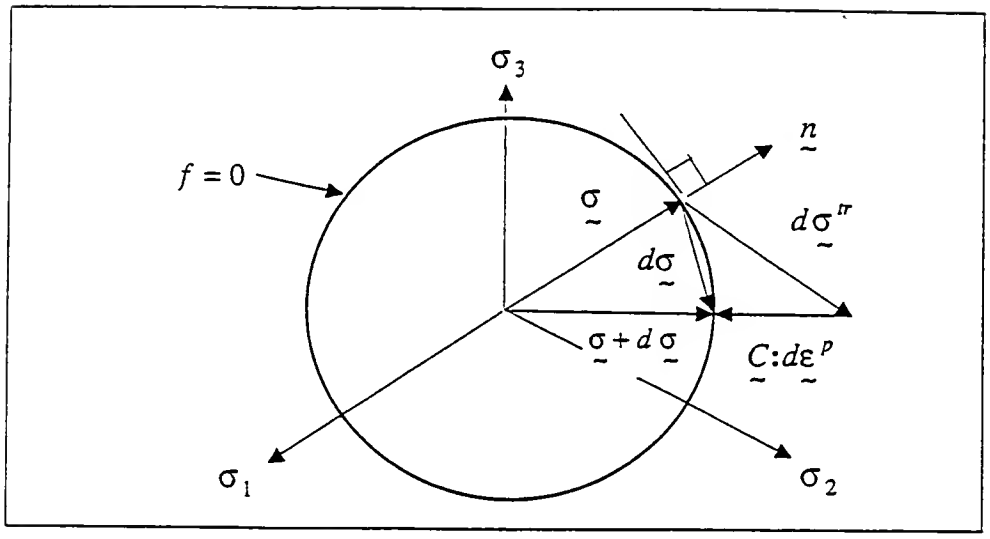

Figure 4.2 Schematic Consistency Condition

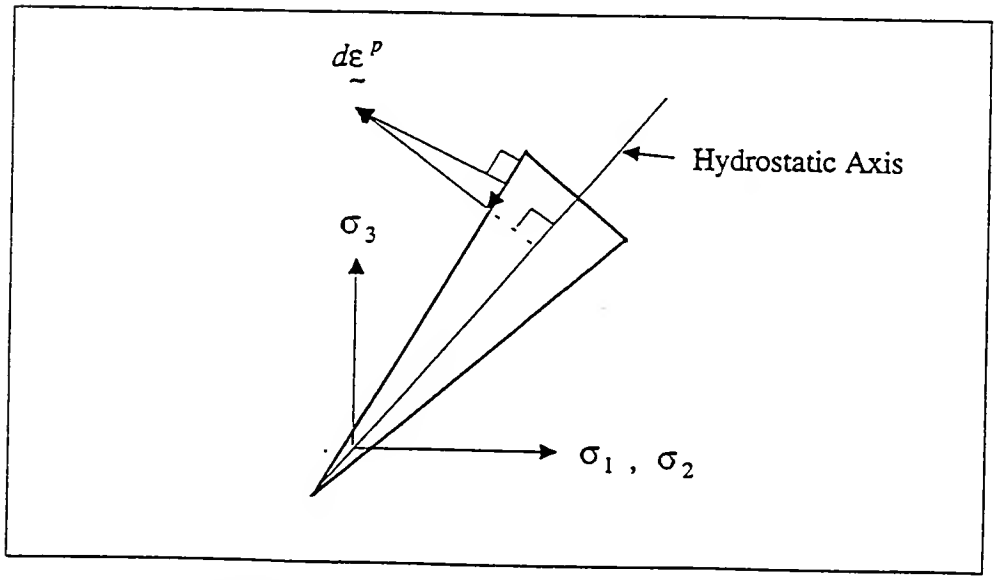

Figure 4.3 The Dilation of Drucker-Prager Yield Function with Associated Flow Rule

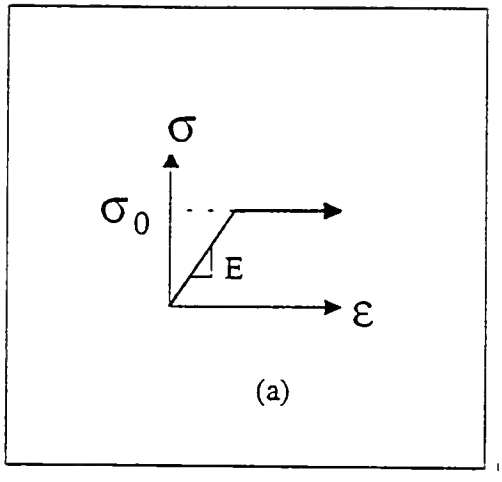

Figure 4.4 Idealized Stress-Strain Curves: Elastic-Perfectly Plastic Model

CHAPTER 5. THREE DIMENSIONAL SOLID ELEMENTS

In chapter 2, general formulations of the co-rotational approach, the principle of virtual work and the general equations of motion are described. Specific formulations for a plane solid element are given.

For a three-dimensional solid element, the formulations for the rotation, virtual work, and the equations are the same. The only differences are:

(a) there are three rotational matrices for the transformation between the convected coordinates and the global coordinates,

(b) there are three displacement components and six independent stress and strain components.

5.1 Rotational Matrix for 3D Solid Elements

The transformation between a set of convected coordinates $(\hat{x},\hat{y},\hat{z})$ and the global coordinates (x,y,z) requires three angles, known as the Euler angles in calculus. Computation of the angles is the same as given in Eq. (2.5). For each plane, (xy), (y,z) and (zx), a rotational angle can be calculated which yields Euler angles (θ_z, θ_x, θ_y). Then, the rotation matrix for the transformation is

$$\underset{\sim}{R} = \underset{\sim}{R}_x \underset{\sim}{R}_y \underset{\sim}{R}_z \tag{5.1}$$

where $\underset{\sim}{R}_x$, $\underset{\sim}{R}_y$, *and* $\underset{\sim}{R}_z$ are rotation matrix for a plane solid.

For example,

$$\underset{\sim}{R}_z = \begin{vmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{5.2}$$

where c= cos θ_x and s= sin θ_x

5.2 Isoparametric Elements for 3D Solid

An eight-node 3D isoparametric element is considered. There are three nodal displacements (u_i,v_i,w_i) at each node. Hence, the element has 24 degrees of freedom. The shape functions are

$$\underset{\sim}{\hat{x}} = \sum_{i=1}^{8} N_i(s,t,r)\underset{\sim}{\hat{x}}_i \tag{5.3}$$

$$\underset{\sim}{\hat{u}} = \sum_{i=1}^{8} N_i(s,t,r)\underset{\sim}{\hat{u}}_i \tag{5.4}$$

$$where\ \underset{\sim}{\hat{x}} = \begin{vmatrix} \hat{x} \\ \hat{u} \\ \hat{z} \end{vmatrix}\ and\ \underset{\sim}{\hat{u}} = \begin{vmatrix} \hat{u} \\ \hat{v} \\ \hat{w} \end{vmatrix}$$

(s, t, r) are three natural coordinates. The eight shape functions have the form

$$N_i = \frac{1}{8}(1 + s_i s)(1 + t_i t)(1 + r_i r) \qquad i = 1,2,3,...,8 \tag{5.5}$$

where (s_i, t_i, r_i) are position values of the node in the natural coordinates. The six strain

components can be calculated by

$$\begin{Vmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Vmatrix} = \begin{vmatrix} \frac{\partial}{\partial \hat{x}} & 0 & 0 \\ 0 & \frac{\partial}{\partial \hat{y}} & 0 \\ 0 & 0 & \frac{\partial}{\partial \hat{z}} \\ \frac{\partial}{\partial \hat{y}} & \frac{\partial}{\partial \hat{x}} & 0 \\ 0 & \frac{\partial}{\partial \hat{z}} & \frac{\partial}{\partial \hat{y}} \\ \frac{\partial}{\partial \hat{z}} & 0 & \frac{\partial}{\partial \hat{x}} \end{vmatrix} \begin{Vmatrix} \hat{u} \\ \hat{v} \\ \hat{w} \end{Vmatrix} \tag{5.6}$$

written in the natural coordinates, the derivate $\frac{\partial f}{\partial \hat{x}}$ for example, becomes

$$\frac{\partial f}{\partial \hat{x}} = \frac{1}{|J|} \begin{vmatrix} f_s & y_s & z_s \\ f_t & y_t & z_t \\ f_r & y_r & z_r \end{vmatrix} \tag{5.7}$$

where $f_s = \frac{\partial f}{\partial s}$, etc.

The Jacobian |J| is

$$|J| = \begin{vmatrix} x_s & y_s & z_s \\ x_t & y_t & z_t \\ x_r & y_r & z_r \end{vmatrix} \tag{5.8}$$

Fig. 5.1 shows the sketch of an eight-node isoparametric element.

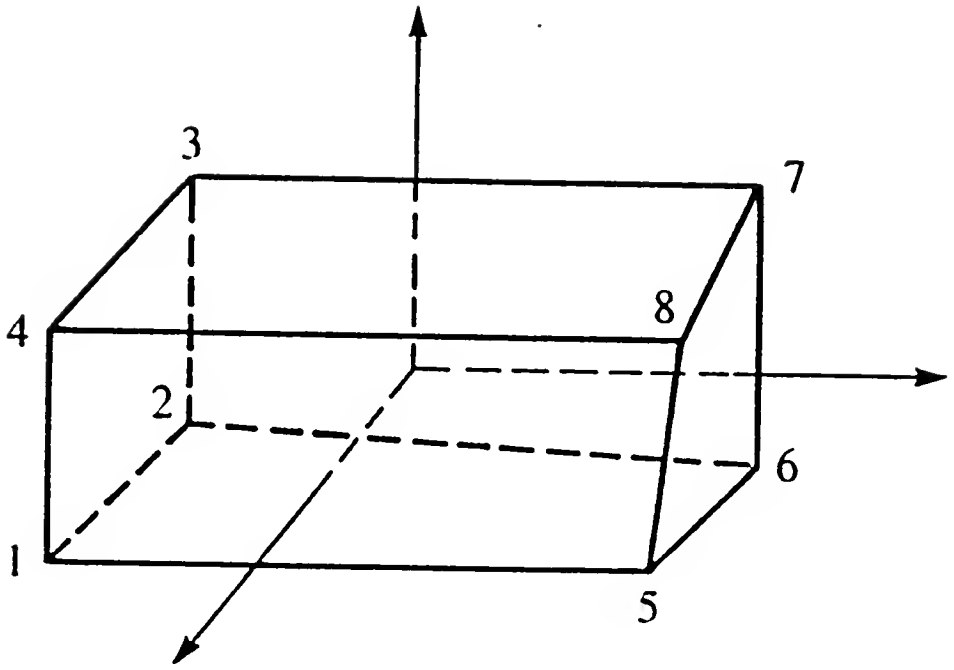

Figure 5.1 An Eight-Node Three-Dimensional Isoparametric Solid Element

Conclusions

1. The algorithm which is based on a vector formulation of finite element for the equations of motion, an explicit time integration technique for the solutions, and a co-rotational formulation for the large displacement seem to work well. The programs are efficient for transient dynamic problems. However, static solutions are also shown to be reliable and accurate.

2. The programs based on this algorithm are compact, efficient, and flexible for expansions.

3. A well-known general purpose commercial code ANSYS is used to verify the codes.

4. Six verification problems are considered for the three dimensional code and five for the two dimensional code.

5. Material models included in the codes are elastic, elastic-plastic (three versions), and viscoelastic. Modifications of the material library are convenient.

6. The load library has included general time history, ground acceleration, and harmonic inputs. By specifying the time histories, pulse loadings and ramp loadings for static solutions can be considered.

LIST OF REFERENCES

Stoner, T.W., Bhatti, M.A., Kim, S.S., Kou, J.K., Milinas-Vega, I. and Amhof, B., "Dynamic Simulation Methods for Evaluating Motor Vehicle and Roadway Design and Resolving Policy Issues," Iowa University Public Policy Center, Iowa City, Iowa, January 1990.

Koubaa, A. and Krauthammer, T., "Numerical Assessment of Three- Dimensional Rigid Pavement Joints Under Impact Loads," MN/RD-91/03, Minnesota Department of Transportation, Maplewood, Minnesota, August 1990.

Bala, K.V. and Kennedy, C.K., "The Structural Evaluation of Flexible Highway Pavements Using the Deflectograph," Proceedings, 2nd International Conference on the Bearing Capacity of Roads and Airfields, Plymouth, England, September 16-18, 1986.

Barksdale, R.D., "Compressive Stress Pluse Tunes in Flexible Pavement for Use in Dynamic Testing," Highway Research Record No. 345, Highway Research Board, Washington, D.C., January 1971, pp. 32-44.

Paterson, W.D.O., "Design Study of Asphalt Membrane - Overlay for Concrete Runway Pavement," Transportation Research Record No. 930, Transportation Research Board, Washington, D.C., January 1983, pp. 1-11.

Zaghloul, S. and White, T.D., "Use of a Three Dimensional-Dynamic Finite Element Program for Analysis of Flexible Pavements," submitted to Transportation Research Board, National Academy of Sciences, Washington, D.C., January 1993.

Zaghloul, S. and WHITE, T. D., "Load Equivalency Factors for Asphalt Pavements," submitted to the Transportation Research Board, National Academy of Sciences, Washington, D. C. January 1993

Bathe, K. J., Finite Element procedures in Engineering Analysis, Prentice-Hall, Englewood Cliffs, N. J., 1982.

Bathe, K. J., Ramm, E., and Wilson, E. L., "Finite Ellement Formulations for Large Deformation Dynamic Analysis," Internaltional Journal of Numerical Methods in Engineering, Vol. 9, 1975.

Key, S. W., "Finite Element Porocedure for the Large Deformation Dynamic Response of Axisymmetric Solids," Journal of Computational Methods in Applied Mechanics and Engineering, Vol.4, 1974.

Oden, J. T. and Key, S. W., "Analysis of Static Nonlinear Response by Explicit Time Integration," International Journal of Numerical Methods in Engineering, Vol. 7, 1973.

Kennedy, J. M., Belytcshko, and Schoebele, D. F., STRAW - A Nonlinear Fluid-Structural and Thermomechanical Finite Element Program, ANL/RAS 85-8, Argonne National Laboratory, Argonne, Illinois, 1985.

Hallquist, J. O., Theoretical Manual for DYNA3D, Lawrence Livermore Laboratory Report No. UCID -19401, 1982.

Saha, N. and Ting, E. C., "Large Displacement Dynamic Analysis of Space Frames," CE-STR-83-5, Civil Engineering, Purdue University 1983.

Saha, N., "Plasticity-Based Modeling and Analysis of Concrete Structures," Ph.D. Dissertation, Purdue University 1983.

Labbane, M., "Computational Failure Analysis of Reinforced Concrete Plate Assemblage," Ph.D. Dissertation, Purdue University 1991.

Labbane, M. and Ting, E. C., "Elasto-Plastic and Viscoelastic Analysis of Impulsively Loaded Mild Steel Plates," Purdue Civil Engineering Report CE-STR-91-17, 1991.

Ting, E. C. and Chen, W. F., " A Finite Element Approach for Creep and Viscoelastic Buckling of General Plate and Shell Structures," Purdue Civil Engineering Report CE-STR-82- 16, 1982.

Ting, E. C.," Applications of A Plastic-Fracture Model to Concrete Structures," Purdue Civil Engineering Report CE-STR-87-9, 1987.

Rice, D. L. and Ting, E. C., "Large Displacement Transient Analysis of Flexible Structures," Purdue Civil Engineering Report CE-STR-91-37, 1991.

Rice, D. L. and Ting, E. C., " Fragmentation Algorithm for Finite Element Failure Simulation and Analysis," Purdue Civil Engineering Report CE-STR-92-21, 1992.

Borja, R. I., Lee, S. R., and Seed, R. B., "Numerical Simulation of Excavation in Elastoplastic Soils," International Journal of Analytical Methods in Geomechanics, Vol. 13, pp. 231-249, 1989.

Desai, C. S. and Sirivardane, H. J., "Constitutive Laws for Engineering Materials with Emphasis on Geologic Materials," Prentice-Hall, Inc., 1984.

Logan, D. L., "A First Course in the Finite Element Method," PWS-Kent Publishing Co., 1992.

Malvern, L. E., "Introduction to the mechanics of A Continuous Medium," Prentice-Hall, Inc., 1969.

Appendix 1

- A summary of the programs
- Using Microsoft Excel to prepare input data and to plot output data
- User's Manual for a two dimensional finite element program Solid2D (S2DP)
- User's Manual for a three dimensional finite element program Soild3D (S3DP)
- Fortran Source Code for Solid2D
- Fortran Source Code for Solid3D

A SUMMARY OF THE PROGRAMS: Solid2D and Solid3D

1. Load library

The current versions have essentially programmed for three types of forces applied at the nodes.

a. An arbitrary time history of the force

b. A ground acceleration input

c. A sinusoidal force with fixed frequency and amplitude.

Currently, it is set up for 6 different types of force input or 6 types of acceleration input. They can be applied at 30 different nodes. For the sinusoidal force, it can be applied at 10 nodes. The user may input 500 time increments for the time histories of acceleration and forces. These numbers can easily be increased by changing the dimension statements in one subroutine "readata".

Pulse impact forces and variable step loadings are prescribed by the force values at different time increments. To obtain static solutions, force input is carried out by prescribing load history as an one-step time history, a multi-step load history or a ramp load.

2. Boundary and Initial conditions

Displacement constraints are prescribed at the nodes.

By default, the initial velocities the initial displacements are zero. That is, the structure is at rest initially. The programs allow the initial displacements and initial velocities to be prescribed. Currently, they may be prescribed at 10 different nodes. To increase the number, merely change the dimension sizes in the subroutine "readata".

3. Body Forces

Gravity acceleration can be input.

4. Material Library

To retain maximum flexibility for the future development, the material data inputs are not classified according to each individual material model. Thus, material models can easily be combined to represent emerging developments in material modeling. For each material type, all the material data can be made available.

Current material inputs are: Mass density, Young's modulus, Poisson's ratio,
Tensile strength (uniaxial tensile yield stress),
Cohesion and Internal angle of friction (for soil only),
Tangent modulus (slope of the stress and strain curve after yielding),
Hardening rule (assuming linear work-hardening), and
Relaxation time (assuming Maxwell model).

The material models available are:

1. linear elastic material
2. linear viscoelastic material of Maxwell type
3. elastic-plastic material with associated flow rule assuming Mises Criterion
4. elastic-plastic material with associated flow rule assuming Drucker-Prager Criterion
5. elastic-plastic material with non-associated flow rule
   (Drucker-Prager for yielding and Mises for hardening)

For all the elastic-plastic materials, there is a choice of kinematical hardening rule, isotropic hardening rule, and mixed hardening rule (characterized by a factor **beta**

as the percentage of kinematical rule).

With minor modifications, bi-linear elastic material, nonlinear elastic material, complex viscoelastic models, nonlinear creep models, and viscoelastic-plastic material are readily to be implemented.

Using Microsoft Excel to prepare input data and to plot output data

How to create the Input Data File

In case of small and simple finite element mesh, DOS editor is preferred to use to create input data file. However, if a larger or complicate mesh, the MS Excel is more preferable that the previous one. In the next following section, the procedure of creating input data file by MS Excel is illustrated.

1. Follow the Input Guide. and start from card 1 (each data number occupying one cell in MS Excel)

2. When you get to card 4(coordinate of each node), you can just enter information for the first node and use copy command to do the other nodes

3. Following the same procedure for generating the element mesh(card 5).

4. For card 6 to card 21, enter the value of each data.

5. Save the input data file in the DOS format under 's2dp.dat --- for solid2d and s3dp.dat --- for solid3d'.

How to Plot the Results in the MS Excel

1. Open the output file or 's2dp.out or s3dp.out' in the MS Excel.

2. By selecting file type as 'fixed width' and click 'next'

3. Select 'column break line' for each particular data set and click 'next'.

4. Select 'data column format' as 'general' and then click 'finish'

5. Select the sets of data to be plotted by shading the interested area.

6. Click Chart Wizard and locate your plotted area.

7. Select the chart type(recommend the XY scatter or Line).

8. Select the format for the chart

9. Following the command on the screen to select the chart set up.

10. Click Finish.

Solid2D User Manual

and

Input Data Guide

```
*************************************************************************
*                                                                       *
*                    INPUT GUIDE(for Solid2D program)                   *
* --------------------------------------------------------------------- *
*                                                                       *
* Sub. dynamic :                                                        *
*  1. head (20a4)                                                       *
*     ------------                                                      *
*      head    : problem description ( 80 characters )                  *
*                                                                       *
*  2. iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha               *
*     ---------------------------------------------------               *
*      iprob   : no. of time-steps skipped between outputs              *
*      nnd     : total no. of nodal points                              *
*      nel     : total no. of elements                                  *
*      nummat  : total no. of different materials or sections used      *
*      numout  : total no. of output records requested --- each d.o.f.  *
*                of each node's d,v,a,sigma forms each output record.   *
*      ndof    : degree of freedom per node (= 2 )                      *
*      maxstp  : max. time steps or cycles of calculation               *
*      delta   : time increment (sec), must be less than critical value *
*      alpha   : coeff. of mass damping                                 *
*      toler   : tolerance limit as a switch to stop running program    *
*                when the increment of displacement is smaller than it  *
*      gravity: acceleration of gravity                                 *
*                (1) no gravity load imposed if gravity=0.              *
*                (2) Gravity load is imposed in the initial condition.  *
*                (3) Displacement due to the gravity load imposed is    *
*                    NOT initialized.                                   *
*                                                                       *
*  3. iacc,iforce,inital,imesh,iplane                                   *
*     ------------------------------------                              *
*      iacc    : index for ground acceleration time-history input       *
*                 0 = no acc.     1 = x-dir      2 = y-dir             *
*      iforce  : index for force function                               *
*                 0 = no external force function                        *
*                 1 = arbitrary shape force function                    *
*                 2 = type of F= f*sin(wt)                              *
*                 3 = type of F= f*cos(wt)                              *
*      inital  : index for initial condition                            *
*                 0 : d=0 v=0       2 : d=0 v                           *
*                 1 : d   v=0       3 : d   v                           *
*      imesh   : index for printing out nodal coor. & element data      *
*                 0 =  not print out                                    *
*                 1 =      print out                                    *
*      iplane  : index for plane stress = 1                             *
*                       or plane strain = 2                             *
*                                                                       *
* --------------------------------------------------------------------- *
*                                                                       *
* Sub. readata :                                                        *
*  4. n,xc(n),yc(n),(ifixx(i),i=1,2)                                    *
*     -----------------------------                                     *
*         n     : node no.                                              *
*      xc(n)    : x - coord. of node n                                  *
*      yc(n)    : y - coord. of node n                                  *
*      ifixx(1): x - translation  B.C.  ( 0 = free ,  1 = fixed )       *
*      ifixx(2): y - translation  B.C.                                  *
*                                                                       *
*  5. n,node(1),node(2), ..... ,node(7),node(8),knt                     *
```

```
*     ------------------------------------------------                  *
*       n     : element no.                                             *
*      node(1) : node 1 for element n    ==>   node(4) : node 4         *
*      node(5) : material property no.                                  *
*      node(6) : row no.    for element n                               *
*      node(7) : column no. for element n                               *
*      node(8) : element condition (=1) (for checking Jacobian use )    *
*        knt   : no. of node number increment used for automatic        *
*                element generation (for horizontal & vertical dir.)    *
*                                                                       *
*  6. k,e(1,k),e(2,k),.....,e(11,k)                                     *
*     ------------------------------------------                        *
*          k  : material group no.                                      *
*      e(1,k) : material type no.          (= 1--metal;= 2 - soil)      *
*      e(2,k) : mass density               ( = rho )                    *
*      e(3,k) : young's modulus            ( = eyng = E )               *
*      e(4,k) : poisson ratio              ( = poisson  )               *
*      e(5,k) : tensile strength of metal  ( = ft )                     *
*      e(6,k) : cohesion for soil          ( = cohesion )               *
*  7. e(7,k),e(8,k),.....,e(12,k)                                       *
*     ------------------------------------------                        *
*      e(7,k) : friction angle for soil    ( = phiangle )               *
*      e(8,k) : material model             ( = 0 for Linear Elastic     *
*                                            = 1 for Viscoelastic       *
*                                            = 2 for von Mises,         *
*                                            = 3 for Drucker-Prager)    *
*      e(9,k)  : tangent modulus            ( = eynt = Et )             *
*      e(10,k) : hardening rules           ( = beta  ) (0 =< beta =< 1)*
*                                  ( = 0 : kinematic ; = 1 : isotropic )*
*      e(11,k) : plate thickness                                        *
*      e(12,k) : tau for Viscoelastic                                   *
*                                                                       *
*                                                                       *
*     ( skip 8,9,10 ,if iacc = 0 ) ---------------------------------- *
*  8. nacc,npnts                                                        *
*     -----------                                                       *
*      nacc  :  total no. of different ground acceleration history      *
*      npnts :  no. of time-acc. pairs in each acc. history             *
*                                                                       *
*  9.   g    :  gravity acceleration (unit must be consistent)          *
*     ------                                                            *
*  10. ta(j,i),aa(j,i) ---j=1,npnts  i=1,nacc                           *
*     ----------------    --------------------                          *
*      ta(j,i) : time  for acc.                                         *
*      aa(j,i) : value for acc.                                         *
*                                                                       *
*     ( skip 11 -- 16 ,if iforce = 0 ) ------------------------------- *
*     ( skip 11 -- 14 ,if iforce .ne. 1 ) ---------------------------- *
* 11. numif,nnaf                                                       *
*     -----------                                                       *
*      numif :  total no. of impact force history                       *
*      nnaf  :  total no. of nodes applied by arbitrary shape impact    *
*               force function                                          *
*                                                                       *
* 12. kfpnts(i) : total no. of time-force pairs in a force function     *
*     ---------   history  [ 1< kfpnts(i) =< 500 ]                      *
*                                                                       *
* 13. tf(j,i),ff(j,i) --- j=1,jf  i=1,numif  jf=kfpnts(i)              *
*     ---------------    --------------------------------              *
*      tf(j,i) : time  for force                                        *
```

```
*      ff(j,i) : value for force                                          *
*                                                                         *
* 14. jnode(i),jdir(i),jaf(i) --- i=1,nnaf                                *
*     ------------------------    ---------                               *
*      jnode(i) : node no. where impact force is applied                  *
*       jdir(i) : d.o.f. corresponding to which this force is applied     *
*                 1 = x-dir     2 = y-dir                                 *
*        jaf(i) : time-force history no.                                  *
*                                                                         *
* 15. nnaf  : total no. of nodes applied by sinusoidal impact force       *
*     -----   function                                                    *
*                                                                         *
* 16. jnode(i),jdir(i),f(i),omega(i) --- i=i,nnaf                         *
*     -------------------------------    ---------                        *
*      jnode(i): node no. where impact force is applied                   *
*      jdir(i) : d.o.f. corresponding to which this force is applied      *
*                  1 = x-dir     2 = y-dir                                *
*         f(i) : amplitude of this sinusoidal force  F= f*sin(wt)         *
*      omega(i): frequency     "       "             F= f*cos(wt)         *
*                                                                         *
*     ( skip 17 -- 20 ,if inital = 0 ) ------------------------------     *
* 17. ndisi : total no. of displacement type I.C.                         *
*     ------                                                              *
* 18. ndnod(i),kdis(i),disi(i) --- i=1,ndisi                              *
*     -------------------------                                           *
*      ndnod(i): node no. where displ type I.C. is applied                *
*       kdis(i): d.o.f. corresponding to which this I.C. is applied       *
*                  1 = x-dir     2 = y-dir                                *
*       disi(i): value of initial displacement                            *
*                                                                         *
* 19. nveli : total no. of velocity type I.C.                             *
*     ------                                                              *
* 20. nvnod(i),kvel(i),veli(i) --- i=1,nveli                              *
*     -------------------------                                           *
*      nvnod(i): node no. where velicity type I.C. is applied             *
*       kvel(i): d.o.f. corresponding to which this I.C. is applied       *
*                  1 = x-dir     2 = y-dir                                *
*       veli(i): value of initial velocity                                *
*                                                                         *
* 21. kout(1),kout(2),kout(3) --- do loop i=1,numout                      *
*     ------------------------    -------------------                     *
*      kout(1) : node no. for which response output is requested          *
*      kout(2) :  = 0   for displacement request                          *
*                 = 1   for velocity     request                          *
*                 = 2   for acceleration request                          *
*                 = 3   for stresses     request                          *
*      kout(3) : global dof corresponding to which output is requested    *
*            1 = x-dir     2 = y-dir     3 = xy-plane                     *
*                                      (for shear stress only)            *
*************************************************************************
*                           Limitation                                    *
* ------------------------------------------------------------------- *
*  1. ta(500,6),aa(500,6) --- only for 6-type ground accelerations      *
*                             & 500 time-acc. pairs for each type       *
*  2. tf(500,6),ff(500,6) --- only for 6-type force functions           *
*                             & 500 time-force pairs for each type      *
*  3. jnode(30),jdir(30),jaf(30) --- only for 30 positions applied by   *
*                             each type force function                  *
*  4. jnode(30),jdir(30),f(10),omega(10) --- only for 10 positions      *
*                             applied by sinusoidal force function      *
```

```
*  5. ndnod(10),kdis(10),disi(10) --- only for 10 positions applied    *
*                              by initial displacement                 *
*  6. nvnod(10),kvel(10),veli(10) --- only for 10 positions applied    *
*                              by initial velocity                     *
*                                                                      *
*  If the limitations were violated , please change the dimension of   *
*  above variables in sub." readata ".                                 *
*                                                                      *
************************************************************************
************************************************************************
*                              Note                                    *
* ------------------------------------------------------------------ *
*  There are three files in this program:                              *
*         1. Input data file             ====>     's2dp.dat'         *
*         2. Print out of input file     ====>     's2dp.in'          *
*         3. Output(result) file         ====>     's2dp.out'         *
************************************************************************
```

Solid3DUser Manual

and

Input Data Guide

```
*************************************************************************
*                                                                       *
*                          INPUT GUIDE(for solid3D)                     *
* --------------------------------------------------------------------- *
*                                                                       *
* Sub. dynamic :                                                        *
*  1. head (20a4)                                                       *
*     ------------                                                      *
*      head    : problem description ( 80 characters )                  *
*                                                                       *
*  2. iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha               *
*     ---------------------------------------------------               *
*      iprob   : no. of time-steps skipped between outputs              *
*      nnd     : total no. of nodal points                              *
*      nel     : total no. of elements                                  *
*      nummat  : total no. of different materials or sections used      *
*      numout  : total no. of output records requested --- each d.o.f.  *
*                of each node's d,v,a,sigma forms each output record.   *
*      ndof    : degree of freedom per node (= 3 )                      *
*      maxstp  : max. time steps or cycles of calculation               *
*      delta   : time increment (sec), must be less than critical value *
*      alpha   : coeff. of mass damping                                 *
*      toler   : tolerance limit as a switch to stop running program    *
*                when the increment of displacement is smaller than it  *
*      gravity: acceleration of gravity                                 *
*               (1) no gravity load imposed if gravity=0.               *
*               (2) Gravity load is imposed in the initial condition.   *
*               (3) Displacement due to the gravity load imposed is     *
*                   NOT initialized.                                    *
*                                                                       *
*  3. iacc,iforce,inital,sideL,imesh,iplane                             *
*     ------------------------------------                              *
*      iacc    : index for ground acceleration time-history input       *
*                 0 = no acc.     1 = x-dir   2 = y-dir   3 = z-dir     *
*      iforce  : index for force function                               *
*                 0 = no external force function                        *
*                 1 = arbitrary shape force function                    *
*                 2 = type of F= f*sin(wt)                              *
*                 3 = type of F= f*cos(wt)                              *
*      inital  : index for initial condition                            *
*                 0 : d=0 v=0       2 : d=0 v                           *
*                 1 : d   v=0       3 : d   v                           *
*      imesh   : index for printing out nodal coor. & element data      *
*                 0 =  not print out                                    *
*                 1 =      print out                                    *
* --------------------------------------------------------------------- *
*                                                                       *
* Sub. readata :                                                        *
*  4. n,xc(n),yc(n),zc(n),(ifixx(i),i=1,3)                              *
*     -------------------------------                                   *
*         n     : node no.                                              *
*      xc(n)    : x - coord. of node n                                  *
*      yc(n)    : y - coord. of node n                                  *
*      zc(n)    : z - coord. of node n                                  *
*      ifixx(1): x - translation  B.C.  ( 0 = free ,  1 = fixed )       *
*      ifixx(2): y - translation  B.C.                                  *
*      ifixx(2): z - translation  B.C.                                  *
*                                                                       *
*  5. n,node(1),node(2), ..... ,node(11),node(12),knt                   *
*     ----------------------------------------------                    *
```

```
*        n     : element no.                                                *
*      node(1) : node 1 for element n   ==>   node(8) : node 8             *
*      node(9) : material property no.                                     *
*      node(10) : row no.    for element n                                 *
*      node(11) : column no. for element n                                 *
*      node(12) : element condition (=1) (for checking Jacobian use )      *
*        knt    : no. of node number increment used for automatic          *
*                 element generation (for horizontal & vertical dir.)     *
*                                                                         *
*  6. k,e(1,k),e(2,k),.....,e(6,k)                                        *
*     ------------------------------------------                          *
*          k  : material group no.                                        *
*      e(1,k) : material type no.          (= 1--metal;= 2 - soil)        *
*      e(2,k) : mass density               ( = rho )                      *
*      e(3,k) : young's modulus            ( = eyng = E )                 *
*      e(4,k) : poisson ratio              ( = poisson  )                 *
*      e(5,k) : tensile strength of metal  ( = ft )                       *
*      e(6,k) : cohesion for soil          ( = cohesion )                 *
*  7. e(7,k),e(8,k),.....,e(10,k)                                         *
*     ------------------------------------------                          *
*      e(7,k) : friction angle for soil    ( = phiangle )                 *
*      e(8,k) : Material Model             ( = 0 for Linear Elastic       *
*                                            = 1 for Viscolastic          *
*                                            = 2 for von Mises,           *
*                                            = 3 for Drucker-Prager)      *
*  if (e(8,k) = 0) ---> e(9..11,k) = 0                                    *
*  if (e(8,k) = 1) ---> e(9..10,k) = 0                                    *
*      e(9,k) : tangent modulus            ( = eynt = Et )                *
*      e(10,k) : hardening rules           ( = beta  ) (0 =< beta =< 1)*
*                                  ( = 0 : kinematic ; = 1 : isotropic )*
*      e(11,k) : Tau for Viscoelastic                                     *
*                                                                         *
*     ( skip 8,9,10 ,if iacc = 0 ) ---------------------------------     *
*  8. nacc,npnts                                                          *
*     -----------                                                         *
*      nacc  :  total no. of different ground acceleration history        *
*      npnts :  no. of time-acc. pairs in each acc. history               *
*                                                                         *
*  9.   g    :  gravity acceleration (unit must be consistent)            *
*     ------                                                              *
* 10. ta(j,i),aa(j,i) --- j=1,npnts  i=1,nacc                             *
*     ----------------    --------------------                            *
*      ta(j,i) : time  for acc.                                           *
*      aa(j,i) : value for acc.                                           *
*                                                                         *
*     ( skip 11 -- 16 ,if iforce = 0 ) ------------------------------     *
* 11. numif,nnaf                                                          *
*     -----------                                                         *
*      numif :  total no. of impact force history                         *
*      nnaf  :  total no. of nodes applied by arbitrary shape impact      *
*               force function                                            *
*                                                                         *
* 12. kfpnts(i) : total no. of time-force pairs in a force function       *
*     ---------   history  [ 1< kfpnts(i) =< 500 ]                        *
*                                                                         *
* 13. tf(j,i),ff(j,i) --- j=1,jf  i=1,numif  jf=kfpnts(i)                 *
*     ----------------    --------------------------------                *
*      tf(j,i) : time  for force                                          *
*      ff(j,i) : value for force                                          *
*                                                                         *
```

```
* 14. jnode(i),jdir(i),jaf(i) --- i=1,nnaf                              *
*     ------------------------    ---------                             *
*      jnode(i) : node no. where impact force is applied                *
*       jdir(i) : d.o.f. corresponding to which this force is applied   *
*            1 = x-dir     2 = y-dir    3 = z-dir                      *
*        jaf(i) : time-force history no.                                *
*                                                                       *
* 15. nnaf  : total no. of nodes applied by sinusoidal impact force     *
*     -----   function                                                  *
*                                                                       *
* 16. jnode(i),jdir(i),f(i),omega(i) --- i=i,nnaf                       *
*     -------------------------------    ---------                      *
*      jnode(i): node no. where impact force is applied                 *
*      jdir(i) : d.o.f. corresponding to which this force is applied    *
*            1 = x-dir     2 = y-dir    3 = z-dir                      *
*          f(i) : amplitude of this sinusoidal force  F= f*sin(wt)      *
*      omega(i): frequency     "      "               F= f*cos(wt)      *
*                                                                       *
*     ( skip 17 -- 20 ,if inital = 0 ) ------------------------------- *
* 17. ndisi : total no. of displacement type I.C.                       *
*     ------                                                            *
* 18. ndnod(i),kdis(i),disi(i) --- i=1,ndisi                            *
*     -------------------------                                         *
*      ndnod(i): node no. where displ type I.C. is applied              *
*       kdis(i): d.o.f. corresponding to which this I.C. is applied     *
*            1 = x-dir     2 = y-dir    3 = z-dir                      *
*       disi(i): value of initial displacement                          *
*                                                                       *
* 19. nveli : total no. of velocity type I.C.                           *
*     ------                                                            *
* 20. nvnod(i),kvel(i),veli(i) --- i=1,nveli                            *
*     -------------------------                                         *
*      nvnod(i): node no. where velicity type I.C. is applied           *
*       kvel(i): d.o.f. corresponding to which this I.C. is applied     *
*            1 = x-dir     2 = y-dir    3 = z-dir                      *
*       veli(i): value of initial velocity                              *
*                                                                       *
* 21. kout(1),kout(2),kout(3) --- do loop i=1,numout                    *
*     ------------------------    -------------------                   *
*      kout(1) : node no. for which response output is requested        *
*      kout(2) :  = 0   for displacement request                        *
*                 = 1   for velocity     request                       *
*                 = 2   for acceleration request                       *
*                 = 3   for stresses     request                       *
*      kout(3) : global dof corresponding to which output is requested  *
*             1 = x-dir     2 = y-dir     3 = z-dir                     *
*             4 = xy-plane  5 = yz-plane  6 = zx-plane                  *
************************************************************************
*                            Limitation                                *
* ---------------------------------------------------------------------- *
*  1. ta(500,6),aa(500,6) --- only for 6-type ground accelerations      *
*                             & 500 time-acc. pairs for each type       *
*  2. tf(500,6),ff(500,6) --- only for 6-type force functions           *
*                             & 500 time-force pairs for each type      *
*  3. jnode(30),jdir(30),jaf(30) --- only for 30 positions applied by   *
*                             each type force function                  *
*  4. jnode(30),jdir(30),f(10),omega(10) --- only for 10 positions      *
*                             applied by sinusoidal force function      *
*  5. ndnod(10),kdis(10),disi(10) --- only for 10 positions applied     *
*                             by initial displacement                   *
```

```
*  6. nvnod(10),kvel(10),veli(10) --- only for 10 positions applied    *
*                              by initial velocity                     *
*                                                                      *
*  If the limitations were violated , please change the dimension of   *
*  above variables in sub." readata ".                                 *
*                                                                      *
************************************************************************
************************************************************************
*                              Note                                    *
* ------------------------------------------------------------------ * *
*  There are three files in this program:                              *
*         1. Input data file              ====>     's3dp.dat'        *
*         2. Print out of input file      ====>     's3dp.in'         *
*         3. Output(result) file          ====>     's3dp.out'        *
************************************************************************
```

Solid2D Source Code

(S2DP.FOR)

```
************************************************************************
*                                                                      *
*   Program for Dynamic Plane Solid Problem ( solid2D )                *
*                                                                      *
************************************************************************
*                                                                      *
*  This program is for analyzing a 2-D dynamic problem and developed   *
*  on the basis of :                                                   *
*   (1) Traditionally Co-Rotational Approach                           *
*   (2) Explicit Time Integration Method (central difference)          *
*   (3) Lumped Mass Modeling                                           *
*   (4) 4-node Solid Isoparametric Element                             *
*   (5) 4-point Gaussian Integration                                   *
*   (6) Elastic-lenear work-hardening Model                            *
*   (7) Viscoelastic Model                                             *
*   (8) von Mises yield criterion                                      *
*   (9) Drucker-Prager yield criterion                                 *
*                                                                      *
*                                          Tatsana Nilaward            *
*                                          Purdue University           *
*                                          03/07/96                    *
*                                                                      *
*                                                                      *
************************************************************************
*                                                                      *
*   Program for Dynamic Plane Solid Problem                            *
*              (Solid2d  = S2DP )                                      *
************************************************************************
c
      program solid2D
      implicit real*8 (a-h,o-z)
      dimension ar(30000)
      maxq = 30000
c  create the input and output filenames ( extension part of filename --
c  --.dat & .out ,will be created automatically )
c
      open (5, file='s2dp.dat')
      open (6, file='s2dp.in')
      open (7, file='s2dp.out')

      call dynamic (ar,maxq)
c
      print *, 'TOTALLY COMPLETE ! '
      close (unit=5)
      close (unit=6)
      close (unit=7)
c
      stop
      end
************************************************************************
*                                                                      *
*  set index no. of each variable by dynamic allocation method         *
*  read in control parameters  &  call main subroutines                *
*                                                                      *
************************************************************************
c
      subroutine dynamic (ar,maxq)
c
      implicit real*8 (a-h,o-z)
      dimension ar(maxq)
```

```
      dimension head(20)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
      common /box 5/ maxcyc,maxout,maxmat
c
      do 100 i=1,maxq
  100 ar(i) = 0.0
c
      read(5,130)  head
      write(6,140) head
      write(6,150)
c
c  iprob --- is number of time-steps skipped between output
c
      read(5,*)    iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha
     *              ,toler,gravity
c
      write(6,160) iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha
     *              ,toler,gravity
c
      read(5,*)    iacc,iforce,inita,imesh,iplane
      write(6,170) iacc,iforce,inital,imesh,iplane
c
c  dynamic allocation ------------------
c
      meq    = nnd*ndof
      maxdof = meq
      maxel  = nel
      maxmat = nummat
      maxnod = nnd
      npint  = 1
      nxmass = npint+meq
      na     = nxmass+meq
      nv     = na+meq
      nd     = nv+meq
      nxc    = nd+meq
      nyc    = nxc+maxnod
      nforce = nyc+maxnod
      nifix  = nforce+meq
c
c  element node connectivity
c
      nnode  = nifix+meq
c
c  material properties
c
      nem    = nnode+8*maxel
c
c  output record
c
      maxcyc = maxstp
      maxout = numout
c
      if (maxstp .eq. 0) maxcyc = 1
      if (numout .eq. 0) maxout = 1
c
      nkout  = nem  +12*maxmat
      ndp    = nkout + 3*maxout
      ndn    = ndp + meq
c
```

```
      nsigmaP = ndn + meq
      nsigmaN = nsigmaP + 4*6*nel
      nepslonP = nsigmaN + 4*6*nel
      nepslonN = nepslonP + 4*6*nel
      nPLalphaP = nepslonN + 4*6*nel
      nPLalphaN = nPLalphaP + 4*6*nel
      nPLrP = nPLalphaN + 4*6*nel
      nPLrN = nPLrP + 4*nel
      nelplas = nPLrN + 4*nel
      nelpout = nelplas + 4*nel
      ns1 = nelpout + 4*nel
      ns2 = ns1 + nnd
      ns3 = ns2 + nnd
      ns4 = ns3 + nnd
      nproutv = ns4 + nnd
      maxindex  = nproutv + maxout
c
      if ( maxindex .gt. maxq ) then
       print *, ' There is not enough dimension available. '
       print *, '  ==> Please increase no. in ar(...) & maxq=... .'
      stop
      endif
c
c
      call readata (nnd,nel,nummat,numout,iacc,ndof,ar(nnode),iforce,
     +              imesh,ar(nxc),ar(nyc),ar(nifix),ar(nem),ar(nkout),
     +              inital)
c
      print *, 'call readata -- complete!'
c
c
      call bmass (nel,nnd,ndof,ar(nem),ar(nnode),ar(nxc),ar(nyc)
     +           ,ar(nxmass))
c
      print *, 'call bmass ---- complete!'
c
      time=0.0
c
c
c
      call esolv (time,nel,nnd,ndof,iacc,numout,iforce,ar(nxmass),
     +          ar(nforce),ar(npint),ar(nifix),ar(nd),ar(nv),ar(na),
     +          ar(nxc),ar(nyc),ar(nnode),ar(nem),ar(nkout),maxstp,
     +          ar(ndn),ar(ndp),inital,meq,ar(nsigmaP),ar(nsigmaN),
     +          ar(nepslonP),ar(nepslonN),ar(nPLalphaP),ar(nPLalphaN),
     +          ar(nPLrP),ar(nPLrN),ar(nelplas),ar(nelpout),ar(nproutv),
     +          ar(ns1),ar(ns2),ar(ns3),ar(ns4),iplane)
c

      print *, 'call esolv ---- complete! '
c
c
  130 format (20a4)
  140 format (/2x,'card  1',5x,20a4)
  150 format (1x,80('-'))
  160 format (2x,'card  2',5x,'parameter card',/,
     +        15x,'no of time-steps skipped between outputs =',i6,/,
     +        15x,'number of nodes          =',i10,/,
     +        15x,'number of elements       =',i10,/,
     +        15x,'number of materials      =',i10,/,
```

```
     +        15x,'number of output req    =',i10,/,
     +        15x,'no. of d.o.f/node       =',i10,/,
     +        15x,'no. of time steps       =',i10,/,
     +        15x,'time increment          =',e10.3,/,
     +        15x,'coeff of mass damping   =',e10.3,/,
     +        15x,'tolerance limit         =',e10.3,/,
     +        15x,'acceleration of gravity =',f10.5,/)
  170 format (2x,'card  3',5x,'index card',/,
     +        15x,'index for accel.      =',i10,/,
     +        15x,'index for force       =',i10,/,
     +        15x,'index for I. C.       =',i10,/,
     +        15x,'index for mesh output(1) or not(0)       =',i4,/,
     +        15x,'index for plane stress(1) or strain(2)   =',i4)
c
      return
      end
c
*****************************************************************************
*                                                                           *
*  calaulate translational mass of all nodes in x-dir & y-dir               *
*  and form Mass-matrix --- all at one time                                 *
*                                                                           *
*****************************************************************************
c
      subroutine bmass (nel,nnd,ndof,e,rnode,xc,yc,xmass)
c
      implicit real*8 (a-h,o-z)
      dimension rnode(8,1),xc(1),yc(1),e(12,1),xmass(1)
      dimension xmasc(8)
      common /Gauss/ s(4),t(4),w(4)
c
c  calculate mass for each node
c
      meq = nnd*ndof
c
      do 100 i=1,meq
  100 xmass(i) = 0.0
c
c  set Gauss points and weights ( w=1 for 4 point Gauss Quadrature )
c
      const = dsqrt(3.0d0)
      s(1) = -1.0/const
      t(1) = -1.0/const
      w(1) =  1.0
      s(2) =  1.0/const
      t(2) = -1.0/const
      w(2) =  1.0
      s(3) =  1.0/const
      t(3) =  1.0/const
      w(3) =  1.0
      s(4) = -1.0/const
      t(4) =  1.0/const
      w(4) =  1.0
c
c  loop through all element
c
      do 200 i=1,nel
c
c  set node no. and node coord. for element
c
```

```
      n1 = int(rnode(1,i))
      n2 = int(rnode(2,i))
      n3 = int(rnode(3,i))
      n4 = int(rnode(4,i))
      x1 = xc(n1)
      x2 = xc(n2)
      x3 = xc(n3)
      x4 = xc(n4)
      y1 = yc(n1)
      y2 = yc(n2)
      y3 = yc(n3)
      y4 = yc(n4)
c
c  material properties for element ( b=thickness, rho=mass density )
c
      mtyp = int(rnode(5,i))
       rho = e(2,mtyp)
       b = e(11,mtyp)
c
      do 300 j=1,8
  300 xmasc(j) = 0.0
c
c  set shape function Ni
c
      do 400 k=1,4
      shpf1 = (1.0-s(k)) * (1.0-t(k)) / 4.0
      shpf2 = (1.0+s(k)) * (1.0-t(k)) / 4.0
      shpf3 = (1.0+s(k)) * (1.0+t(k)) / 4.0
      shpf4 = (1.0-s(k)) * (1.0+t(k)) / 4.0
c
c  derivatives of shape function w.r.t. s,t
c
      s1ds = -(1.0-t(k)) / 4.0
      s2ds =  (1.0-t(k)) / 4.0
      s3ds =  (1.0+t(k)) / 4.0
      s4ds = -(1.0+t(k)) / 4.0
      s1dt = -(1.0-s(k)) / 4.0
      s2dt = -(1.0+s(k)) / 4.0
      s3dt =  (1.0+s(k)) / 4.0
      s4dt =  (1.0-s(k)) / 4.0
c
c  derivatives of global cood. x,y  w.r.t. s,t
c
      xs = s1ds*x1 + s2ds*x2 + s3ds*x3 + s4ds*x4
      ys = s1ds*y1 + s2ds*y2 + s3ds*y3 + s4ds*y4
      xt = s1dt*x1 + s2dt*x2 + s3dt*x3 + s4dt*x4
      yt = s1dt*y1 + s2dt*y2 + s3dt*y3 + s4dt*y4
c
      detJ = xs*yt - ys*xt
c
c  compute translational masses --- xmasc(8)
c
      xmasc(1) = xmasc(1) + b*rho*w(k)*shpf1*detJ
      xmasc(2) = xmasc(2) + b*rho*w(k)*shpf1*detJ
      xmasc(3) = xmasc(3) + b*rho*w(k)*shpf2*detJ
      xmasc(4) = xmasc(4) + b*rho*w(k)*shpf2*detJ
      xmasc(5) = xmasc(5) + b*rho*w(k)*shpf3*detJ
      xmasc(6) = xmasc(6) + b*rho*w(k)*shpf3*detJ
      xmasc(7) = xmasc(7) + b*rho*w(k)*shpf4*detJ
      xmasc(8) = xmasc(8) + b*rho*w(k)*shpf4*detJ
```

```
  400 continue
c
c  calculate index of mass storage
c
      m1 = (n1-1)*ndof
      m2 = (n2-1)*ndof
      m3 = (n3-1)*ndof
      m4 = (n4-1)*ndof
c
c  assemble to global mass matrix --- xmass(8)
c
      xmass(m1+1) = xmass(m1+1) + xmasc(1)
      xmass(m1+2) = xmass(m1+2) + xmasc(2)
      xmass(m2+1) = xmass(m2+1) + xmasc(3)
      xmass(m2+2) = xmass(m2+2) + xmasc(4)
      xmass(m3+1) = xmass(m3+1) + xmasc(5)
      xmass(m3+2) = xmass(m3+2) + xmasc(6)
      xmass(m4+1) = xmass(m4+1) + xmasc(7)
      xmass(m4+2) = xmass(m4+2) + xmasc(8)
  200 continue
c
      return
      end
**************************************************************************
*                                                                        *
*  compute elastic constitutive coefficients under  3D condition         *
*                                                                        *
*                                                                        *
**************************************************************************
c
      subroutine elastd3d (eyng,poisson,ed)
c
      implicit real*8 (a-h,o-z)
      dimension ed(6,6)
c
      do 50 i=1,6
      do 60 j=1,6
       ed(i,j) = 0.d0
   60 continue
   50 continue
c
      G  = eyng/(2.d0*(1.d0+poisson))
      dla  = poisson*eyng/((1.d0+poisson)*(1.d0-2.d0*poisson))
c
      ed(1,1) = 2.d0*G + dla
      ed(2,2) = ed(1,1)
      ed(3,3) = ed(1,1)
      ed(4,4) = G
      ed(5,5) = ed(4,4)
      ed(6,6) = ed(4,4)
      ed(1,2) = dla
      ed(2,1) = ed(1,2)
      ed(3,1) = ed(1,2)
      ed(3,2) = ed(1,2)
      ed(1,3) = ed(1,2)
      ed(2,3) = ed(1,2)
c
      return
      end
c
```

```
c
**********************************************************************
*                                                                    *
*  Solve Eq. of Motion by Displacement-based Central Difference Method *
*                                                                    *
**********************************************************************
c
      subroutine esolv (time,nel,nnd,ndof,iacc,numout,iforce,xmass,
     +                  force,pint,rifix,d,v,a,xc,yc,rnode,e,rkout,
     +                  maxstp,dn,dp,inital,meq,sigmaP,sigmaN,epslonP,
     +                  epslonN,PLalphaP,PLalphaN,PLrP,PLrN,elplas,
     +                  elpout,proutv,s1,s2,s3,s4,iplane)
c
      implicit real*8 (a-h,o-z)
      dimension xmass(1),force(1),pint(1),rnode(8,1),rkout(3,1)
      dimension d(1),v(1),a(1),xc(1),yc(1),e(12,1),rifix(1)
      dimension dn(1),dp(1),ag(3),elplas(1),elpout(1)
      dimension sigmaP(1),sigmaN(1),epslonP(1),epslonN(1),proutv(1)
      dimension PLalphaP(1),PLalphaN(1),PLrP(1),PLrN(1)
      dimension s1(1),s2(1),s3(1),s4(1)

      common /box 1/ iprob,delta,alpha,toler,gravity
      common /box 4/ npnts,numif,nnaf,ndisi,nveli,
     +               kfpnts(6),jnode(30),jdir(30),jaf(30),
     +               ndnod(10),kdis(10),nvnod(10),kvel(10)
      common /box 4a/g,disi(10),veli(10),f(10),omega(10),
     +               ff(500,6),ta(500,6),tf(500,6),aa(500,6)
c  define initial condition that time is within the range
c     -- see subroutine finter in detail
c
      noyes = 0
c
      do 100 i=1,meq
        d(i) = 0.0
        v(i) = 0.0
        a(i) = 0.0
  100 force(i) = 0.0
c
      if (inital .eq. 0) go to 50
      if (inital .eq. 2) go to 60
c
c  set index no. of d,v, and put initial value into them
c
      do 65 i=1,ndisi
      ind = (ndnod(i)-1)*ndof+kdis(i)
      d(ind) = disi(i)
   65 continue
c
      if (inital .ne. 3) go to 50
c
   60 continue
      do 70 i=1,nveli
      inv = (nvnod(i)-1)*ndof+kvel(i)
      v(inv) = veli(i)
   70 continue
c
c  compute displacement (nstep=0) before first step displ.(nstep=1)
c  by Central Difference Method
c
   50 continue
```

```
      do 75 i=1,meq
      dn(i) = d(i)-delta*v(i)+0.5*delta*delta*a(i)
   75 continue
c
      nstep = 0
      nskip = 0
c
c
  160 do 120 i=1,meq
  120 force(i) = 0.
c
cc impose gravity load
c
      do 140 i=2,meq,2
  140 force(i) = -1.*gravity*xmass(i)
c
c@d--20jan1996--cshih
c
cc impose impact force
c
c  set index no. of nodal d.o.f. applied by impact force
c  compute external force at that time step by linear interpolation
c
      if (iforce .eq. 0) go to 167
c
      do 165 n=1,nnaf
      imn = (jnode(n)-1)*ndof+jdir(n)
      if (iforce .ne. 1) go to 163
      ma = jaf(n)
      mb = kfpnts(ma)
      call finter (ff(1,ma),tf(1,ma),mb,time,pht,noyes)
      force(imn) = pht + force(imn)
      go to 165
c
  163 if(iforce.eq.2)force(imn)=f(n)*dsin(omega(n)*time)+force(imn)
      if(iforce.eq.3)force(imn)=f(n)*dcos(omega(n)*time)+force(imn)
  165 continue
c
  167 continue
c
cc impose ground acceleration
c
      if (iacc .eq. 0) go to 25
c
      do 5 i=1,3
    5 ag(i) = 0.0
c
c  compute acceleration at that time step by linear interpolation
c
      if (iacc .lt. 4) go to 13
      if (iacc .eq. 4) go to 12
c
      do 11 j=1,3
      call finter (aa(1,j),ta(1,j),npnts,time,pht,noyes)
      ag(j) = pht
   11 continue
      go to 14
c
   12 continue
      call finter (aa(1,1),ta(1,1),npnts,time,pht,noyes)
```

```
      ag(1) = pht
      call finter (aa(1,2),ta(1,2),npnts,time,pht,noyes)
      ag(3) = pht
      go to 14
c
   13 call finter (aa(1,1),ta(1,1),npnts,time,pht,noyes)
      ag(iacc) = pht
c
c  compute external force by using  d'Alembert principle to account for
c  inertia force
c
   14 do 20 i=1,nnd
      ii = (i-1)*ndof
      do 21 j=ii+1,ii+2
      force(j) = force(j)-xmass(j)*ag(j-ii)*g
   21 continue
   20 continue
   25 continue
c
c initialized index for average stress output
c
c
      do 998 ig = 1,numout
      proutv(ig) = 0.d0
  998 continue
c
      do 999 io = 1,nnd
      s1(io) = 0.0d0
      s2(io) = 0.0d0
      s3(io) = 0.0d0
      s4(io) = 0.0d0
  999 continue
c
      do 991 ki = 1,4*nel
      elplas(ki) = 0.0d0
      elpout(ki) = 0.0d0
  991 continue
c
c  compute element internal nodal forces ( pint )
c
      do 22 i=1,meq
   22 pint(i)=0.
c
      do 23 i=1,nel
      neo = int(rnode(8,i))
      if (neo.lt.0  .or. neo.eq.11) goto 23
      call fintiso8 (i,ndof,xc,yc,rnode,e,d,pint,sigmaP,sigmaN,
     +               epslonP,epslonN,PLalphaP,PLalphaN,PLrP,PLrN,
     +               elplas,s1,s2,s3,s4,iplane,time)
   23 continue
c
      do 360 i=1,nel
      do 360 k=1,4
      n = (i-1)*4
      do 370 j=1,6
       m = (n+k-1)*6+j
       sigmaP(m) = sigmaN(m)
       epslonP(m) = epslonN(m)
       PLalphaP(m) = PLalphaN(m)
  370 continue
```

```
        PLrP(n+k) = PLrN(n+k)
  360 continue
c
c  compute displacement of next time step --------------`dp(j)
c       and velocity & acceleration of this time step --- v(j) & a(j)
c  by Displacement-based Central Difference Method
c
        do 170 i=1,nnd
        m = (i-1)*ndof
        do 171 j=m+1,m+2
        if (int(rifix(j)) .eq. 1) go to 171
        if (xmass(j) .le. 0.0) go to 171
         a1 = 2.0+alpha*delta
         a2 = 2.0-alpha*delta
         c1 = (2.0/a1)*delta*delta
         c2 = 4.0/a1
         c3 = a2/a1
         c4 = (force(j)-pint(j))/xmass(j)
        dp(j) = c1*c4+c2*d(j)-c3*dn(j)
         v(j) = (dp(j)-dn(j))/(2.0*delta)
         a(j) = (dp(j)-2.0*d(j)+dn(j))/(delta*delta)
  171 continue
  170 continue
c
c
c record plastic element number
c
        if (nskip .eq. iprob) then
        write(6,*) 'nstep=',nstep
        ij = 0
        do 556 i=1,4*nel
        if (elplas(i) .lt. 1.0) go to 556
        ij = ij + 1
        elpout(ij) = elplas(i)
  556 continue
        write (6,2021)
 2021 format(/,'  Plastic element no [element no.Gauss point no] =')
        write (6,2031) (elpout(i),i=1,ij)
 2031 format(8(3x,f5.1))
        if (ij .eq. 0) write(6,2032)
 2032 format('      NONE')
        endif
c
c  print out response results requested every "iprob" step
c
        if (nskip .ne. iprob) go to 195
        if (numout .ne. 0) then
        call prout (numout,rkout,d,v,a,s1,s2,s3,s4,ndof,nstep,
     +             time,proutv)
         nskip = 0
        endif
  195 if (nstep .ge. maxstp) go to 225
c
c  check whether the increment of each nodal displacement is less than
c  the tolerance limit, i.e. TOLER.
c
        if(nstep .lt. 200) go to 198
        ddmax = 0.0d0
        do 197 i=1,meq
           yn = dabs(dp(i)-d(i))
```

```
  197 if (ddmax .lt. yn) ddmax = yn
      if(ddmax .lt. toler) then
       write (*,1100) ddmax
 1100 format (/,' STOP -- displ.increment < tolerance limit',1pd9.3,/)
      go to 225
      endif
  198 continue
c
      do 200 i=1,meq
      dn(i) = d(i)
      d(i)  = dp(i)
  200 continue
c
      nskip = nskip + 1
      nstep = nstep + 1
      time  = time + delta
      go to 160
c
  225 continue
c
c record plastic element numberfor last time step
c
      write(6,*) 'nstep=',nstep
      ij = 0
      do 555 i=1,4*nel
      if (elplas(i) .lt. 1.0) go to 555
      ij = ij + 1
      elpout(ij) = elplas(i)
  555 continue
      write (6,2001)
 2001 format(/,'  Plastic element no =>[Element no.Gauss point no] =')
      write (6,2011) (elpout(i),i=1,ij)
 2011 format(8(3x,f5.1))
      if (ij .eq. 0) write(6,2022)
 2022 format('      NONE')
      print *,'complete!'
c
      return
      end
c



**********************************************************************
*                                                                    *
*  calculate each time step's external force and acceleration by linear*
*  interpolation ---- because of mismatch between the time interval of *
*  force & acc. input data, and the time increment                    *
*                                                                    *
**********************************************************************
c
      subroutine finter (pp,vv,kc,t,pht,noyes)
c
      implicit real*8 (a-h,o-z)
      dimension pp(kc),vv(kc)
c
      if (t.lt.vv(1) .or. t.gt.vv(kc)) then
       if (noyes .eq. 1) go to  900
       write(6,*)' '
       write(6,*)' WARNING -- time out of the range of time-force pairs'
```

```
       write(6,*)'            ZERO FORCE is given at that time.'
       write(6,*)'            the out-of-the-range time is ',t
       write(6,*)' '
       print *, ' '
       print *, ' WARNING -- time out of the range of time-force pairs'
       print *, '            ZERO FORCE is given at that time.'
       print *, '            the out-of-the-range time is ',t
       print *, ' '
       noyes = 1
  900    pht = 0.0
       go to 140
      endif
c
      do 100 l=2,kc
       if ( t .le. vv(l)) go to 120
  100 continue
c
  120 pht = pp(l-1)+(t-vv(l-1))*(pp(l)-pp(l-1))/(vv(l)-vv(l-1))
c
  140 return
c
c@d-22oct1995-c.shih
c
      end
c
*************************************************************************
*                                                                       *
*  compute element internal nodal forces ( pint ) by  ---               *
*  4-node plane isoparametric element ( 4-node quadrilateral element )  *
*                                                                       *
*************************************************************************
c
      subroutine fintiso8 (i,ndof,xc,yc,rnode,e,d,pint,sigmaP,sigmaN,
     +             epslonP,epslonN,PLalphaP,PLalphaN,PLrP,PLrN,elplas,
     +             s1,s2,s3,s4,iplane,time)
c
      implicit real*8 (a-h,o-z)
      dimension xc(1),yc(1),rnode(8,1),e(12,1),d(1),pint(1)
      dimension disl(8),f(8),bmx(3,8),te(3,3)
      dimension epsx(4),epsy(4),epsxy(4)
      dimension ed(6,6),sigma(3),sig(4,3),sigma3d(6)
      dimension Depslon(6),PLalpha(6),elplas(1)
      dimension sigmaP(1),sigmaN(1),epslonP(1),epslonN(1)
      dimension PLalphaP(1),PLalphaN(1),PLrP(1),PLrN(1)
      dimension s1(1),s2(1),s3(1),s4(1)
      common /box 1/ iprob,delta,alpha,toler,gravity
      common /Gauss/ s(4),t(4),w(4)
      data phi/3.1415926535898/
c
c  set node no. for element
c
      n1 = int(rnode(1,i))
      n2 = int(rnode(2,i))
      n3 = int(rnode(3,i))
      n4 = int(rnode(4,i))
c
c  node(8) = 1 ---  element condition ( read in sub"readata" )
c  if node(8) = neo = 11  ---> element has a negative Jacobian
c
      neo= int(rnode(8,i))
```

```
c
c
c
      if (neo.eq.11) go to 13
c
c  set index no. of nodes
c
      n1n = (n1-1)*ndof
      n2n = (n2-1)*ndof
      n3n = (n3-1)*ndof
      n4n = (n4-1)*ndof
c
c  set node coordinates
c
      x1 = xc(n1)
      x2 = xc(n2)
      x3 = xc(n3)
      x4 = xc(n4)
      y1 = yc(n1)
      y2 = yc(n2)
      y3 = yc(n3)
      y4 = yc(n4)
c
c  set node displacement
c
      d1x = d(n1n+1)
      d1y = d(n1n+2)
      d2x = d(n2n+1)
      d2y = d(n2n+2)
      d3x = d(n3n+1)
      d3y = d(n3n+2)
      d4x = d(n4n+1)
      d4y = d(n4n+2)
c
c defomed coordinates
c
      x1d = x1 + d1x
      y1d = y1 + d1y
      x2d = x2 + d2x
      y2d = y2 + d2y
      x3d = x3 + d3x
      y3d = y3 + d3y
      x4d = x4 + d4x
      y4d = y4 + d4y
c
c check distored element
c
      A123 = dabs(x1d*y2d+x2d*y3d+x3d*y1d-x1d*y3d-x2d*y1d-x3d*y2d)
      A134 = dabs(x1d*y3d+x3d*y4d+x4d*y1d-x1d*y4d-x3d*y1d-x4d*y3d)
      A234 = dabs(x2d*y3d+x3d*y4d+x4d*y2d-x2d*y4d-x3d*y2d-x4d*y3d)
      A124 = dabs(x1d*y2d+x2d*y4d+x4d*y1d-x1d*y4d-x2d*y1d-x4d*y2d)
c
      if ( A123 .lt. 1.0d-07 ) go to 8
      if ( A134 .lt. 1.0d-07 ) go to 8
      if ( A234 .lt. 1.0d-07 ) go to 8
      if ( A124 .lt. 1.0d-07 ) go to 8
      if ( dabs(A123+A134-A234-A124) .gt. 1.0d-07 ) go to 8
      go to 5
c
    8 write(6,10) i
```

```
   10 format(1x,'-- Distorted element no. = ',i5)
      rnode(8,i) = 11.0
      go to 13
c
    5 continue
c
c  find rotational angle of element,
c  based on the lines composite with centroid and each nodal point
c  the coordinates of the centroid of element = ( xcd, ycd )
c
      xcdu = (x1 + x2 + x3 + x4) / 4.d0
      ycdu = (y1 + y2 + y3 + y4) / 4.d0
      xcdd = (x1d + x2d + x3d + x4d) / 4.d0
      ycdd = (y1d + y2d + y3d + y4d) / 4.d0
c
      call thetafind (xcdu,ycdu,xcdd,ycdd,x1d,y1d,x1,y1,theta1)
      call thetafind (xcdu,ycdu,xcdd,ycdd,x2d,y2d,x2,y2,theta2)
      call thetafind (xcdu,ycdu,xcdd,ycdd,x3d,y3d,x3,y3,theta3)
      call thetafind (xcdu,ycdu,xcdd,ycdd,x4d,y4d,x4,y4,theta4)
      theta = (theta1 + theta2 + theta3 + theta4) / 4.d0
      if (abs(theta) .le. 1.0d-5) theta = 0.0d0
 50   if (theta .lt. 2.0d+0*phi) go to 60
      theta = theta - 2.0d+0*phi
      go to 50
 60   continue
c
c  form transformation matrix --- global to local
c
      c11 =  dcos( theta )
      c12 =  dsin( theta )
      c21 = -c12
      c22 =  c11
c
c  compute local node coord. and displ. ( rotation from global to local
)
c
      x11 = c11*x1 + c12*y1
      y11 = c21*x1 + c22*y1
      x12 = c11*x2 + c12*y2
      y12 = c21*x2 + c22*y2
      x13 = c11*x3 + c12*y3
      y13 = c21*x3 + c22*y3
      x14 = c11*x4 + c12*y4
      y14 = c21*x4 + c22*y4
c
      disl(1) = c11*d1x + c12*d1y
      disl(2) = c21*d1x + c22*d1y
      disl(3) = c11*d2x + c12*d2y
      disl(4) = c21*d2x + c22*d2y
      disl(5) = c11*d3x + c12*d3y
      disl(6) = c21*d3x + c22*d3y
      disl(7) = c11*d4x + c12*d4y
      disl(8) = c21*d4x + c22*d4y
c
      do 100 j=1,4
      epsx(j)  = 0.0d0
      epsy(j)  = 0.0d0
      epsxy(j) = 0.0d0
      f(2*j-1) = 0.0d0
  100 f(2*j  ) = 0.0d0
```

```
c
c  coeff. of 4 point Gauss Quadrature have been transmitted from :
c     common /Gauss/ s(4),t(4),w(4)   ---- in sub."bmass"
c  set shape function Ni
c
      do 200 k=1,4
      shpf1 = (1.d0-s(k)) * (1.d0-t(k)) / 4.d0
      shpf2 = (1.d0+s(k)) * (1.d0-t(k)) / 4.d0
      shpf3 = (1.d0+s(k)) * (1.d0+t(k)) / 4.d0
      shpf4 = (1.d0-s(k)) * (1.d0+t(k)) / 4.d0
c
c  derivatives of shape function w.r.t. s,t
c
      s1ds = -(1.d0-t(k)) / 4.d0
      s2ds =  (1.d0-t(k)) / 4.d0
      s3ds =  (1.d0+t(k)) / 4.d0
      s4ds = -(1.d0+t(k)) / 4.d0
      s1dt = -(1.d0-s(k)) / 4.d0
      s2dt = -(1.d0+s(k)) / 4.d0
      s3dt =  (1.d0+s(k)) / 4.d0
      s4dt =  (1.d0-s(k)) / 4.d0
c
c  derivatives of global cood. x,y  w.r.t. s,t
c
      xs = s1ds*xl1 + s2ds*xl2 + s3ds*xl3 + s4ds*xl4
      ys = s1ds*yl1 + s2ds*yl2 + s3ds*yl3 + s4ds*yl4
      xt = s1dt*xl1 + s2dt*xl2 + s3dt*xl3 + s4dt*xl4
      yt = s1dt*yl1 + s2dt*yl2 + s3dt*yl3 + s4dt*yl4
c
      detJ = xs*yt - ys*xt
c
      if (detJ .le. 0.d0) then
       print *,  '--- Negative Jacobian, ele. no = ', i ,' J= ', detJ
       write(6,*)'--- Negative Jacobian, ele. no = ', i ,' J= ', detJ
       write(6,*)'node no. 1,2,3,4 = ', n1,n2,n3,n4
       write(6,*)'local x= ',xl1,xl2,xl3,xl4,'global x= ',x1,x2,x3,x4
       write(6,*)'local y= ',yl1,yl2,yl3,yl4,'global y= ',y1,y2,y3,y4
       rnode(8,i) = 11.0
       go to 13
      endif
c
c  form B matrix
c
      b1x = yt*s1ds - ys*s1dt
      b1y = xs*s1dt - xt*s1ds
      b2x = yt*s2ds - ys*s2dt
      b2y = xs*s2dt - xt*s2ds
      b3x = yt*s3ds - ys*s3dt
      b3y = xs*s3dt - xt*s3ds
      b4x = yt*s4ds - ys*s4dt
      b4y = xs*s4dt - xt*s4ds
c
      bmx(1,1) = b1x
      bmx(1,2) = 0.d0
      bmx(1,3) = b2x
      bmx(1,4) = 0.d0
      bmx(1,5) = b3x
      bmx(1,6) = 0.d0
      bmx(1,7) = b4x
      bmx(1,8) = 0.d0
```

```
      bmx(2,1) = 0.d0
      bmx(2,2) = b1y
      bmx(2,3) = 0.d0
      bmx(2,4) = b2y
      bmx(2,5) = 0.d0
      bmx(2,6) = b3y
      bmx(2,7) = 0.d0
      bmx(2,8) = b4y
      bmx(3,1) = b1y
      bmx(3,2) = b1x
      bmx(3,3) = b2y
      bmx(3,4) = b2x
      bmx(3,5) = b3y
      bmx(3,6) = b3x
      bmx(3,7) = b4y
      bmx(3,8) = b4x
c
c  compute local element strain
c
      do 300 j=1,8
      epsx(k)  = epsx(k)  + bmx(1,j)*disl(j) / detJ
      epsy(k)  = epsy(k)  + bmx(2,j)*disl(j) / detJ
      epsxy(k) = epsxy(k) + bmx(3,j)*disl(j) / detJ
  300 continue

c
c assign value of new epslon (epslonN(..)
c
c check plane strain problem
c
      if (iplane .eq. 2) then
      m = ((i-1)*4+k-1)*6
      epslonN(m+1) = epsx(k)
      epslonN(m+2) = epsy(k)
      epslonN(m+3) = 0.0d0
      epslonN(m+4) = epsxy(k)
      epslonN(m+5) = 0.0d0
      epslonN(m+6) = 0.0d0
      end if
c
c check plane stress problem
c
      if (iplane .eq. 1) then
      m = ((i-1)*4+k-1)*6
      epslonN(m+1) = epsx(k)
      epslonN(m+2) = epsy(k)
      epslonN(m+3) = -(poisson/(1-poisson))*(epsx(k)+epsy(k))
      epslonN(m+4) = epsxy(k)
      epslonN(m+5) = 0.0d0
      epslonN(m+6) = 0.0d0
      end if
c
c calculate delta epslon
c
      do 352 in = 1,6
       Depslon(in) = epslonN(m+in) - epslonP(m+in)
  352 continue
c
c  material properties for element ( b=thickness, eyng= E )
c
```

```
      mtyp    = int(rnode(5,i))
      mtyp2   = int(e(1,mtyp))
      eyng    = e(3,mtyp)
      if (eyng .le. 0.0d0) goto 13
      poisson = e(4,mtyp)
      ft      = e(5,mtyp)
      coh     = e(6,mtyp)
      phiangle = e(7,mtyp)*phi/180.d0
      kickPL   = int(e(8,mtyp))
      eyngt    = e(9,mtyp)
      beta     = e(10,mtyp)
      b        = e(11,mtyp)
      tau      = e(12,mtyp)
c
c  call constitutive coefficient from subroutine.
c
      call elastd3d (eyng,poisson,ed)
c
      if (kickPL .eq. 1) then
      do 402 m = 1,6
      do 502 n = 1,6
      ed(m,n) = exp(-time/tau)*ed(m,n)
  502 continue
  402 continue
      end if
c
c calculate trial stress(sigma = stresses at gauss pts)
c

      ny = (i-1)*4 + k
      PLr = PLrP(ny)
      do 353 m=1,6
       ny = ((i-1)*4+k-1)*6+m
       sigma3d(m) = sigmaP(ny)
       PLalpha(m) = PLalphaP(ny)
  353 continue
      do 400 m = 1,6
      do 500 n = 1,6
       sigma3d(m) = sigma3d(m) + ed(m,n)*Depslon(n)
  500 continue
  400 continue
c
      sigmean = sigma3d(1)+sigma3d(2)+sigma3d(3)/3.0d0
c
c apply plasticity model : Drucker-Prager yield criterion
c
      if(kickPL .ne. 1)
     +call PLmodel(sigma3d,eyng,poisson,ft,coh,phiangle,kickPL,
     +          eyngt,beta,PLalpha,PLr,sigmean,mtyp2,elplas,i,k)
c
      PLrN((i-1)*4+k) = PLr
      do 800 m=1,6
       ny = ((i-1)*4+k-1)*6+m
       sigmaN(ny) = sigma3d(m)
       PLalphaN(ny) = PLalpha(m)
  800 continue
c
c  transfer stresses from 3d to plane strain 2d condition
c
      sigma(1) = sigma3d(1)
```

```
      sigma(2) = sigma3d(2)
      sigma(3) = sigma3d(4)
c
c  transform local nodal stresses into global
c  --sig(i,k) = sigma{i=x-(1),y-(2),xy-(3); gauss pts(k=1,4)
c
      te(1,1) = c11*c11
      te(1,2) = c12*c12
      te(1,3) = c11*c12
      te(2,1) = te(1,2)
      te(2,2) = te(1,1)
      te(2,3) = -te(1,3)
      te(3,1) = -2.0d0*te(1,3)
      te(3,2) =  2.0d0*te(1,3)
      te(3,3) = te(1,1) - te(1,2)
c
      do 777 mi=1,3
      sig(k,mi) = 0.0d0
      do 810 ni=1,3
        sig(k,mi) = sig(k,mi) + te(ni,mi)*sigma(ni)
  810 continue
  777 continue

c
c  compute local element internal nodal forces
c
      do 600 m=1,8
      do 700 n=1,3
      f(m) = f(m) + b*w(k)*bmx(n,m)*sigma(n)
  700 continue
  600 continue
c
  200 continue
c
c Extrapolate stress from Gauss's pts to node
c and calculate average nodal stress
c
      call stress(sig,n1,n2,n3,n4,s1,s2,s3,s4)
c
c  transform local nodal forces to global
c
      f1x = c11*f(1) + c21*f(2)
      f1y = c12*f(1) + c22*f(2)
      f2x = c11*f(3) + c21*f(4)
      f2y = c12*f(3) + c22*f(4)
      f3x = c11*f(5) + c21*f(6)
      f3y = c12*f(5) + c22*f(6)
      f4x = c11*f(7) + c21*f(8)
      f4y = c12*f(7) + c22*f(8)
c
c  assemble to global nodal force matrix --- pint (internal)
c
      pint(n1n+1) = pint(n1n+1) + f1x
      pint(n1n+2) = pint(n1n+2) + f1y
      pint(n2n+1) = pint(n2n+1) + f2x
      pint(n2n+2) = pint(n2n+2) + f2y
      pint(n3n+1) = pint(n3n+1) + f3x
      pint(n3n+2) = pint(n3n+2) + f3y
      pint(n4n+1) = pint(n4n+1) + f4x
      pint(n4n+2) = pint(n4n+2) + f4y
```

```
c
   13 return
      end
c
*************************************************************************
* Drucker-Prager yield criterion with non-associate flow rule and       *
* linear combination of isotropic and kinematic hardening/softening     *
* Krieq and Key's radial-return algorithm for elastoplastic case        *
*          ( kickPL = 3 for Drucker-Prager)                             *
*          ( beta = 0. for kinematic & =1 for isotropic hardening)      *
*************************************************************************
c
      subroutine PLmodel(sigma3d,eyng,poisson,ft,coh,phiangle,kickPL,
     +            eyngt,beta,PLalpha,PLr,sigmean,mtyp2,elplas,i,k)
c
      implicit real*8 (a-h,o-z)
      dimension PLalpha(1),sigma3d(1),xi(6),elplas(1)
c
      Ep = eyngt/(1.0d0 - eyngt/eyng)
      gshear = eyng/(2.0d0*(1.0d0 + poisson))
      c1 = 2.0d0*gshear + Ep*2.0d0/3.0d0
      c2 = 2.0d0/3.0d0*beta*Ep
      c3 = 2.0d0/3.0d0*(1.0d0-beta)*Ep
      c4 = 2.0d0*gshear
c
c calculate xi-trial
c
      do 10 j=1,6
       xi(j) = sigma3d(j) - PLalpha(j)
   10 continue
c
c calculate deviatoric part of xi
c
      ximean = (xi(1)+xi(2)+xi(3))/3.0d0
c
c calculate the radius of cross section of cone/cylinder
c
      if ((mtyp2 .eq. 2).and.(kickPL .eq. 2)) phiangle = 0.0d0
      if(kickPL .eq. 2) then
       b = 0.0d0
       ak = ft/dsqrt(3.0d0)
      endif
      if (kickPL .eq. 3) then
       b = dsqrt(12.d0)*dsin(phiangle)/(3.0d0+dsin(phiangle))
       ak = dsqrt(12.d0)*(-coh)*dcos(phiangle)/(3.d0+dsin(phiangle))
      endif
       r = dabs(dsqrt(2.d0)*(ak+b*sigmean))
      if (r .gt. PLr) PLr = r
c
      do 20 j =1,3
       xi(j) = xi(j) - ximean
   20 continue
c
c check if elastic
c
      xxi = xi(1)*xi(1)+xi(2)*xi(2)+xi(3)*xi(3)+
     +     2.d0*(xi(4)*xi(4)+xi(5)*xi(5)+xi(6)*xi(6))
      yn = PLr*PLr
      if (xxi .le. yn) go to 100
c
```

```
c Plastic phase: calculate unit normal--N (also store in "xi")
c
      xxi = dsqrt(xxi)
      do 30 j =1,6
       xi(j) = xi(j)/xxi
   30 continue
c
c calculate lambda-tilde ("A")
c
      A = (xxi-PLr)/c1
c
c update stresses
c
      PLr = PLr + c2*A
      t1 = c3*A
      t2 = c4*A
      do 40 j=1,6
       PLalpha(j) = PLalpha(j) + t1*xi(j)
       sigma3d(j) = sigma3d(j) - t2*xi(j)
   40 continue
c
c  record the element number if plastic
c
      j = 4*(i-1)+k
      elplas(j) = dble(i) + 0.1d0*dble(k)
c

  100 return
      end
c
**************************************************************************
*                                                                        *
*  compute element internal nodal streses(sigma) by  ---                 *
*  8-node solid isoparametric element (8-node hexahedral element)        *
*                                                                        *
**************************************************************************
c
      subroutine stress(sig,n1,n2,n3,n4,s1,s2,s3,s4)
c
      implicit real*8 (a-h,o-z)
      dimension x(4),y(4)
      dimension sig(4,3),signodeX(4),signodeY(4),signodeXY(4)
      dimension sN(4,4),s1(1),s2(1),s3(1),s4(1)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
c
c Interpolate stresses from gauss points to nodes
c ( si(idof,io) ; idof=dof, io= local node no. )
c
      constant = dsqrt(3.0d0)
      x(1) = -constant
      y(1) = -constant
c
      x(2) =  constant
      y(2) = -constant
c
      x(3) =  constant
      y(3) =  constant
c
      x(4) = -constant
```

```
      y(4) =  constant
c
      do 203 j = 1,4
      sN(j,1) = (1.d0-x(j))*(1.d0-y(j))/4.d0
      sN(j,2) = (1.d0+x(j))*(1.d0-y(j))/4.d0
      sN(j,3) = (1.d0+x(j))*(1.d0+y(j))/4.d0
      sN(j,4) = (1.d0-x(j))*(1.d0+y(j))/4.d0
  203 continue
c
      do 501 j = 1,4
        signodeX(j) = 0.0d0
        signodeY(j) = 0.0d0
        signodeXY(j) = 0.0d0
         do 502 k = 1,4
           signodeX(j) = signodeX(j) + sN(j,k)*sig(k,1)
           signodeY(j) = signodeY(j) + sN(j,k)*sig(k,2)
           signodeXY(j) = signodeXY(j) + sN(j,k)*sig(k,3)
  502 continue
  501 continue
c
      s1(n1) = s1(n1) + signodeX(1)
      s1(n2) = s1(n2) + signodeX(2)
      s1(n3) = s1(n3) + signodeX(3)
      s1(n4) = s1(n4) + signodeX(4)

c
      s2(n1) = s2(n1) + signodeY(1)
      s2(n2) = s2(n2) + signodeY(2)
      s2(n3) = s2(n3) + signodeY(3)
      s2(n4) = s2(n4) + signodeY(4)

c
      s3(n1) = s3(n1) + signodeXY(1)
      s3(n2) = s3(n2) + signodeXY(2)
      s3(n3) = s3(n3) + signodeXY(3)
      s3(n4) = s3(n4) + signodeXY(4)
c
      s4(n1) = s4(n1) + 1.0d0
      s4(n2) = s4(n2) + 1.0d0
      s4(n3) = s4(n3) + 1.0d0
      s4(n4) = s4(n4) + 1.0d0
   13 return
      end
*****************************************************************************
*                                                                           *
*  print out final results requested at specific points & directions        *
*                                                                           *
*****************************************************************************
c
      subroutine prout (numout,rkout,d,v,a,s1,s2,s3,s4,ndof,
     +                  nstep,time,proutv)
c
c  This subroutine controls output of displacement, velocity,
c  acceleration, and stresses.
c
      implicit real*8 (a-h,o-z)
      dimension d(ndof,1),v(ndof,1),a(ndof,1)
      dimension rkout(3,1),proutv(1),s1(1),s2(1),s3(1),s4(1)
c
      do 100 i2=1,numout
```

```
      node = int(rkout(1,i2))
      idva = int(rkout(2,i2))
      idof = int(rkout(3,i2))
       if (idva .eq. 0) proutv(i2) = d(idof,node)
       if (idva .eq. 1) proutv(i2) = v(idof,node)
       if (idva .eq. 2) proutv(i2) = a(idof,node)
       if (idva .eq. 3) then
c
        if (idof .eq. 1) proutv(i2) = s1(node)/s4(node)
        if (idof .eq. 2) proutv(i2) = s2(node)/s4(node)
        if (idof .eq. 3) proutv(i2) = s3(node)/s4(node)
       end if
  100 continue
       write (*,200) nstep
  200 format(' [',i7,']')
      write(7,300) time,(proutv(i),i=1,numout)
  300 format(1x,'time =',e12.5,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3))
c
      return
      end







************************************************************************
*                                                                      *
*  read in node & element data , material properties                   *
*       acc. & force data, initial condition data, and output request  *
*                                                                      *
************************************************************************
c
      subroutine readata (nnd,nel,nummat,numout,iacc,ndof,rnode,
     +                    iforce,imesh,xc,yc,rifix,e,rkout,inital)
c
      implicit real*8 (a-h,o-z)
      dimension rifix(1),rnode(8,1),rkout(3,1),xc(1),yc(1),e(12,1)
      dimension ifixx(2),node(8),kout(3)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
      common /box 4/ npnts,numif,nnaf,ndisi,nveli,
     +               kfpnts(6),jnode(30),jdir(30),jaf(30),
```

```
     +               ndnod(10),kdis(10),nvnod(10),kvel(10)
      common /box 4a/g,disi(10),veli(10),f(10),omega(10),
     +               ff(500,6),ta(500,6),tf(500,6),aa(500,6)
c
      meq = nnd*ndof
c
      do 100 i=1,meq
  100 rifix(i) = 0.0
c
c  read & write nodal coord. &  B.C.
c  skipped nodes will be automatically generated
c
      write(6,550)
      l = 0
  110 read(5,*)  n,xc(n),yc(n),(ifixx(i),i=1,2)
c
      do 120 j=1,2
      n1 = (n-1)*ndof
      rifix(n1+j) = dble(ifixx(j))
  120 continue
c
      n1 = l+1
c
      if (l .eq. 0) go to 130
c
      dl = n-l
      dxl = (xc(n)-xc(l))/dl
      dyl = (yc(n)-yc(l))/dl
c
  130 continue
      l = l+1
c
      if (n-l) 180,170,150
c
  150 xc(l) = xc(l-1)+dxl
      yc(l) = yc(l-1)+dyl
c
      do 155 j=1,2
      l1 = (l-1)*ndof
      l2 = (l-2)*ndof
      rifix(l1+j) = rifix(l2+j)
  155 continue
c
      if (imesh .eq. 1) then
       write(6,570) l,xc(l),yc(l),(int(rifix(i)),i=l1+1,l1+2)
      endif
      go to 130
c
 170  continue
      if (imesh .eq. 1) then
       write(6,570) n,xc(n),yc(n),(int(rifix(k)),k=n1+1,n1+2)
      endif
      if (nnd-n) 180,190,110
c
  180 continue
      write(6,580) n
      stop
c
  190 continue
c
```

```
c  read & write element connectivity
c
      write(6,600)
      l = 0
  200 read(5,*)  n,node(1),node(2),node(3),node(4),node(5),node(6),
     +             node(7),node(8),knt
c
      do 212 i=1,8
      rnode(i,n) = dble(node(i))
  212 continue
c
      nl = l+1
      if (knt .eq. 0) knt = 1
  245 continue
      l = l+1
      if (n-l) 260,255,250
c
 250  rnode(1,l) = rnode(1,l-1) + dble(knt)
      rnode(2,l) = rnode(2,l-1) + dble(knt)
      rnode(3,l) = rnode(3,l-1) + dble(knt)
      rnode(4,l) = rnode(4,l-1) + dble(knt)
      rnode(5,l) = rnode(5,l-1)
      rnode(6,l) = rnode(6,l-1)
      rnode(7,l) = rnode(7,l-1) + dble(knt)
      rnode(8,l) = rnode(8,l-1)
      go to 245
c
  255 continue
      if (imesh .eq. 1) then
      write(6,620) (k,int(rnode(1,k)),int(rnode(2,k)),int(rnode(3,k)),
     +                int(rnode(4,k)),int(rnode(5,k)),int(rnode(6,k)),
     +                int(rnode(7,k)),int(rnode(8,k)),k=nl,n)
      endif
      if (nel-n) 260,270,200
c
  260 continue
      write(6,630) n
      stop
c
  270 continue
c
c  read & write material properties
c
      do 300 i=1,nummat
      read(5,*)  k,e(1,k),e(2,k),e(3,k),e(4,k),e(5,k)
      read(5,*)  e(6,k),e(7,k),e(8,k),e(9,k),e(10,k),e(11,k),e(12,k)
      write(6,960)
      write(6,970) k,int(e(1,k)),e(2,k),e(3,k),e(4,k),e(5,k)
      write(6,980)
      write(6,990) e(6,k),e(7,k),int(e(8,k)),e(9,k),e(10,k),e(11,k)
  300 continue
c
c  read & write no. of acceleration history, and data of time-acc. pairs
c  in each acc. history
c
      if (iacc .eq. 0) go to 760
c
      read(5,*)    nacc,npnts
      read(5,*)    g
      write(6,1170)
```

```
      write(6,1180) nacc,npnts
c     write(6,1185)
c     write(6,1186) g
c
      do 721 i=1,nacc
      read(5,*)  (ta(j,i),aa(j,i),j=1,npnts)
  721 continue
c
  760 continue
      if (iforce .eq. 0) go to 762
      if (iforce .ne. 1) go to 763
c
c  read & write no. of impact force history, no. of nodes, and data of
c  time-acc. pairs in each acc. history
c
      read(5,*)    numif,nnaf
      write(6,1300)
      write(6,1310)numif,nnaf
c
      if (numif .eq. 0) go to 768
c
      do 755 i=1,numif
      read(5,*)  kfpnts(i)
      jf = kfpnts(i)
      read(5,*)  (tf(j,i),ff(j,i),j=1,jf)
      write(6,1295)
      do 785 j=1,jf
      write(6,1210) i,j,tf(j,i),ff(j,i)
  785 continue
  755 continue
c
  768 continue
      write(6,1315)
c
c  read & write data of position applied by arbitrary shape impact force
c  function ( node, d.o.f., history no. )
c
      do 769 i=1,nnaf
      read(5,*)    jnode(i),jdir(i),jaf(i)
      write(6,1225)jnode(i),jdir(i),jaf(i)
  769 continue
      go to 762
c
c  read & write data of position applied by sinusoidal impact force
c  function ( node, d.o.f., history no. )
c
  763 continue
      write(6,1316)
      read(5,*)  nnaf
c
      do 764 i=1,nnaf
      read(5,*)  jnode(i),jdir(i),f(i),omega(i)
      write(6,1226) jnode(i),jdir(i),f(i),omega(i)
  764 continue
c
  762 continue
      if (inital .eq. 0) go to 765
      if (inital .eq. 2) go to 782
c
c  read & write data of displacement type I.C. ( node, dof, I.C.value )
```

```
c
      read(5,*)  ndisi
      write(6,784)
      do 786 i=1,ndisi
      read(5,*)    ndnod(i),kdis(i),disi(i)
      write(6,783) ndnod(i),kdis(i),disi(i)
  786 continue
c
      if (inital .ne. 3) go to 765
c
c  read & write data of velocity type I.C. ( node, dof, I.C.value )
c
  782 read(5,*)  nveli
      write(6,787)
      do 788 i=1,nveli
      read(5,*)    nvnod(i),kvel(i),veli(i)
      write(6,783) nvnod(i),kvel(i),veli(i)
  788 continue
c
c  read & write data of output record requested (node, d-v-a-type, dof)
c
  765 continue
      if (numout .eq. 0) go to 780
c
      write(6,1228)
      write(7,1228)
c
      do 770 i=1,numout
      read(5,*) (kout(j),j=1,3)
      write(6,1229) i,(kout(j),j=1,3)
      write(7,1229) i,(kout(j),j=1,3)
      do 747 j=1,3
      rkout(j,i) = dble(kout(j))
  747 continue
  770 continue
c
  780 continue
c
c
  550 format (/2x,'card  4',7x,'nodal  point data',/,
     +        10x,'node no.',3x,'x-ordinate',2x,'y-ordinate',
     +        4x,'ifx',4x,'ify')
  570 format (10x,i5,2x,2f12.3,2i8)
  580 format (17x,'nodal point error n =',i5)
c
  600 format(/2x,'card  5     element data'/,10x,'ele. no.',1x,'node-1',
     +        1x,'node-2',1x,'node-3',1x,'node-4',1x,'mat-typ',
     +        ' row-no',' col-no',' ele-cond.')
  620 format (10x,i6,1x,i6,1x,i6,1x,i6,1x,i6,1x,i6,1x,i6,1x,i6,1x,i6,1x,i6)
  630 format (17x,'element number error n =',i5)
c
  783 format (i8,6x,i6,10x,f12.5)
  784 format (5x,'node',5x,'disp-dir',8x,'initial disp',/)
  787 format (5x,'node',5x,'vel-dir',8x,'initial vel',/)
c
 1170 format (/,2x,'card  12',6x,'prescribed acceleration card'/)
 1180 format (10x,'total no. of acceleration histories                ',
     +   i5,/,10x,'total no. of time-acc. pairs in each acc. history =',
     +   i5)
c
```

```
  960 format (/2x,'card  6 & 7    material property data',/,10x,
     +'material  material  mass      Youngs    Poisson   tensile'/10x,
     +'group no. type no.  density   modulus    ratio    strength')
  970 format (11x,i3,7x,i3,3x,e10.4,1x,e10.4,2x,f5.3,3x,e10.4)
  980 format (19x,'cohesion   phi     yield     tangent  hardening',/30x
     +'angle   criterion  modulus    rule    thick(b)')
  990 format (17x,e10.4,2x,f6.2,5x,i2,5x,e9.4,2x,f5.3,4x,f5.3)
c
 1200 format (/,2x,'card  13',6x,'acceleration-history card',/,
     +        17x,'pair no.',8x,'time',9x,'acc'/)
 1210 format (20x,i6,11x,i6,4x,e12.4,4x,e12.4)
 1220 format (/,2x,'card  14',6x,'nodal accele. information card',/,
     +        17x,' node','  dir','  acc'/)
 1225 format (15x,i5,8x,i5,14x,i5)
 1226 format (15x,i5,8x,i5,14x,e10.4,3x,f5.3)
 1228 format (/,2x,'card  21  stress output information card',/,14x,
     + 'seq.    node#    d-(0),v-(1),a-(2),sig-(3)    x(1),y(2),xy(3)')
 1229 format (12x,i4,6x,i4,13x,i4,22x,i4)
c@d
 1295 format (/2x,'card  12 & 13     impact force history card',/,17x,
     +       'force history no.'5x,'pair no.',5x,'time',9x,'iforce')
c
 1300 format (/2x,'card  11     prescribed impact force')
 1310 format (20x,'total no. of impact force history          =',i5,/,
     +        20x,'total no. of nodes applied by impact force =',i5)
 1315 format (/2x,'card  14     nodal impact force information',/,
     +        15x,'node no.    x-(1),y(2)   force history no.')
 1316 format (/2x,'card  14     sinusoidal force information',/,
     +        15x,'node no.    x-(1),y-(2),z-(3)     ampli.     freq.')
c
      return
      end
c
*************************************************************************
*                                                                       *
*   use arc-cos function to find theta value                            *
*          theta = dacos( numerator / denominator )                     *
*                                                                       *
*************************************************************************
c
      subroutine thetafind (xcd1,ycd1,xcd2,ycd2,xd,yd,xu,yu,theta)
c
      implicit real*8 (a-h,o-z)
      data phi/3.1415926535898/
c
      xau = xu - xcd1
      yau = yu - ycd1
c
      xad = xd - xcd2
      yad = yd - ycd2
c
c
c compute the length of undeformed shape
c
      rLu = dsqrt(xau*xau + yau*yau)
c
c compute the length of deformed shape
c
      rLd = dsqrt(xad*xad + yad*yad)
c
```

```
c compute the unit components of undeformed vectors
c
      eux1 = xau/rLu
      euy1 = yau/rLu
c
c compute the unit components of deformed vectors
c
      edx1 = xad/rLd
      edy1 = yad/rLd
c
c compute the rigid body rotation
c xy plane rotation(thetaZ)
c
      Cz = eux1*edy1 - edx1*euy1
      theta = dasin(Cz)
      if ((xu*xd .lt. 0.d0).and.(yu*yd .lt. 0.d0)) then
       if (thetaZ .ge. 0.d0) thetaZ =  phi - thetaZ
       if (thetaZ .lt. 0.d0) thetaZ = -phi - thetaZ
      endif
      return
      end
c
```

Solid3D Source Code

(S3DP.FOR)

```
************************************************************************
*                                                                      *
*   Program for Dynamic of 3 Dimensional Solid Element Problem         *
*                                                                      *
************************************************************************
*                                                                      *
*  This program is for analysis of 3-D dynamic problem and developed   *
*  on the basis of :                                                   *
*   (1) Traditionally Co-Rotational Approach                           *
*   (2) Explicit Time Integration Method (central difference)          *
*   (3) Lumped Mass Modeling                                           *
*   (4) 8-node Solid Isoparametric Element                             *
*   (5) 8-point Gaussian Integration                                   *
*   (6) Elastic-lenear work-hardening Model                            *
*   (7) Viscoelastic Model                                             *
*   (8) von Mises yield criterion                                      *
*   (9) Drucker-Prager yield criterion                                 *
*                                                                      *
*                                          Tatsana Nilaward            *
*                                          Purdue University           *
*                                          04/10/96                    *
*                                                                      *
************************************************************************
c
      program solid3D
      implicit real*8 (a-h,o-z)
      dimension ar(200000)
      maxq = 200000
c
c  read data file and create the input and output filenames
c ( extension part of filename:_.dat, .in, and .out ,will be created
automatically )
c
      open (5, file='s3dp.dat')
      open (6, file='s3dp.in')
      open (7, file='s3dp.out')
c
      call allocate (ar,maxq)
c
      print *, 'TOTALLY COMPLETE ! '
      close (unit=5)
      close (unit=6)
      close (unit=7)
c
      stop
      end
c
************************************************************************
*                                                                      *
*  set index no. of each variable by allocation method                 *
*  read in control parameters  &  call main subroutines                *
*                                                                      *
************************************************************************
c
      subroutine allocate (ar,maxq)
c
      implicit real*8 (a-h,o-z)
      dimension ar(maxq)
      dimension head(20)
      common /box 1/ iprob,delta,alpha,toler,gravity
```

```
      common /box 5/ maxcyc,maxout,maxmat
c
      do 100 i=1,maxq
  100 ar(i) = 0.d0
c
      read(5,130)  head
      write(6,140) head
      write(6,150)
c
c  iprob --- is number of time-steps skipped between output
c
      read(5,*)    iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha
     *             ,toler,gravity
      write(6,160) iprob,nnd,nel,nummat,numout,ndof,maxstp,delta,alpha
     *             ,toler,gravity
      read(5,*)    iacc,iforce,inital,imesh
      write(6,170) iacc,iforce,inital,imesh
c
c  allocation of ar() ------------------
c
      meq    = nnd * ndof
      maxel  = nel
      maxmat = nummat
      maxnod = nnd
      npint  = 1
      nxmass = npint + meq
      na     = nxmass + meq
      nv     = na + meq
      nd     = nv + meq
      nxc    = nd + meq
      nyc    = nxc + maxnod
      nzc    = nyc + maxnod
      nforce = nzc + maxnod
      nifix  = nforce + meq
c
c  element node connectivity
c
      nnode  = nifix+meq
c
c  material properties
c
      nem    = nnode+12*maxel
c
c  output record
c
      maxcyc = maxstp
      maxout = numout
c
      if (maxstp .eq. 0) maxcyc = 1
      if (numout .eq. 0) maxout = 1
c
      nkout  = nem + 11*maxmat
      ndp    = nkout + 3*maxout
      ndn    = ndp + meq
c
      nsigmaP = ndn + meq
      nsigmaN = nsigmaP + 8*6*nel
      nepslonP = nsigmaN + 8*6*nel
      nepslonN = nepslonP + 8*6*nel
      nPLalphaP = nepslonN + 8*6*nel
```

```
      nPLalphaN = nPLalphaP + 8*6*nel
      nPLrP = nPLalphaN + 8*6*nel
      nPLrN = nPLrP + 8*nel
      nelplas = nPLrN +8*nel
      nelpout = nelplas +8*nel
      ns1 = nelpout + 8*nel
      ns2 = ns1 + nnd
      ns3 = ns2 + nnd
      ns4 = ns3 + nnd
      ns5 = ns4 + nnd
      ns6 = ns5 + nnd
      ns7 = ns6 + nnd
      nproutv = ns7 + nnd
      maxindex  = nproutv + maxout
c
      if ( maxindex .gt. maxq ) then
      print *, ' There is not enough dimension available. '
      print *, ' Please increase no. in ar(...) & maxq=',maxindex
      stop
      endif
c
c call subroutines
c
      call readata (nnd,nel,nummat,numout,iacc,ndof,ar(nnode),iforce,
     +              imesh,ar(nxc),ar(nyc),ar(nzc),ar(nifix),ar(nem),
     +              ar(nkout),inital)
      print *, 'call readata -- complete!'
c
      call bmass (nel,nnd,ndof,ar(nem),ar(nnode),ar(nxc),ar(nyc),
     +           ar(nzc),ar(nxmass))
      print *, 'call bmass ---- complete!'
c
      time = 0.d0
c
      call esolv (time,nel,nnd,ndof,iacc,numout,iforce,ar(nxmass),
     +          ar(nforce),ar(npint),ar(nifix),ar(nd),ar(nv),ar(na),
     +          ar(nxc),ar(nyc),ar(nzc),ar(nnode),ar(nem),ar(nkout),
     +          maxstp,ar(ndn),ar(ndp),inital,meq,ar(nsigmaP),
     +          ar(nsigmaN),ar(nepslonP),ar(nepslonN),ar(nPLalphaP),
     +          ar(nPLalphaN),ar(nPLrP),ar(nPLrN),ar(nelplas),
     +          ar(nelpout),ar(nproutv),ar(ns1),ar(ns2),ar(ns3),ar(ns4),
     +          ar(ns5),ar(ns6),ar(ns7))
c
      print *, 'call esolv ---- complete! '
c
c
  130 format (20a4)
  140 format (/2x,'card  1',5x,20a4)
  150 format (1x,80('-'))
  160 format (2x,'card  2',5x,'parameter card',/,
     +        15x,'no of time-steps skipped between outputs =',i6,/,
     +        15x,'number of nodes          =',i10,/,
     +        15x,'number of elements       =',i10,/,
     +        15x,'number of materials      =',i10,/,
     +        15x,'number of output req     =',i10,/,
     +        15x,'no. of d.o.f/node        =',i10,/,
     +        15x,'no. of time steps        =',i10,/,
     +        15x,'time increment           =',e10.3,/,
     +        15x,'coeff of mass damping    =',e10.3,/,
     +        15x,'tolerance limit          =',e10.3,/,
```

```
     +        15x,'acceleration of gravity =',f10.5,/)
  170 format (2x,'card  3',5x,'index card',/,
     +        15x,'index for accel.      =',i10,/,
     +        15x,'index for force       =',i10,/,
     +        15x,'index for I. C.       =',i10,/,
     +        15x,'index for mesh output(1) or not(0) =',i4)
c
      return
      end
*************************************************************************
*                                                                       *
*  calaulate translational mass of all nodes in x, y, and z-dir         *
*  and form Mass-matrix --- all at one time                             *
*                                                                       *
*************************************************************************
c
      subroutine bmass (nel,nnd,ndof,e,rnode,xc,yc,zc,xmass)
c
      implicit real*8 (a-h,o-z)
      dimension rnode(12,1),xc(1),yc(1),zc(1),e(11,1),xmass(1)
c
c  calculate mass for each node
c
      meq = nnd*ndof
c
      do 10 i=1,meq
   10 xmass(i) = 0.d0
c
c  loop through all element
c

      do 20 i=1,nel
c
c  set node no. and node coord. for element
c
      n1 = int(rnode(1,i))
      n2 = int(rnode(2,i))
      n3 = int(rnode(3,i))
      n4 = int(rnode(4,i))
      n5 = int(rnode(5,i))
      n6 = int(rnode(6,i))
      n7 = int(rnode(7,i))
      n8 = int(rnode(8,i))

c
      x1 = xc(n1)
      x2 = xc(n2)
      x3 = xc(n3)
      x4 = xc(n4)
      x5 = xc(n5)
      x6 = xc(n6)
      x7 = xc(n7)
      x8 = xc(n8)
c
      y1 = yc(n1)
      y2 = yc(n2)
      y3 = yc(n3)
      y4 = yc(n4)
      y5 = yc(n5)
      y6 = yc(n6)
```

```
      y7 = yc(n7)
      y8 = yc(n8)
c
      z1 = zc(n1)
      z2 = zc(n2)
      z3 = zc(n3)
      z4 = zc(n4)
      z5 = zc(n5)
      z6 = zc(n6)
      z7 = zc(n7)
      z8 = zc(n8)
c
c  material properties for element ( b=thickness, rho=mass density )
c
      mtyp = int(rnode(9,i))
      rho = e(2,mtyp)
c
c  calculate area of base triangle and height
c  aijk = area of triangle with nodes i,j,and k are vertex
c
      a125 = dabs(0.5d0*((x2*z5+x5*z1+x1*z2)-(x2*z1+x5*z2+x1*z5)))
      a438 = dabs(0.5d0*((x4*z8+x8*z3+x3*z4)-(x4*z3+x8*z4+x3*z8)))
      a265 = dabs(0.5d0*((x5*z6+x6*z2+x2*z5)-(x5*z2+x6*z5+x2*z6)))
      a378 = dabs(0.5d0*((x7*z8+x8*z3+x3*z7)-(x7*z3+x8*z7+x3*z8)))
      h1 = dabs(y4-y1)
      h2 = dabs(y3-y2)
      h3 = dabs(y7-y6)
      h4 = dabs(y8-y5)
c
c  calculate average area and height
c
      aavg1 = (a125+a438)/2.d0
      havg1 = (h1+h2+h4)/3.d0
      aavg2 = (a265+a378)/2.d0
      havg2 = (h2+h3+h4)/3.d0
c
c  compute each half mass of element i th
c
      rmhalf1 = rho*aavg1*havg1
      rmhalf2 = rho*aavg2*havg2
      totmass = rmhalf1+rmhalf2
c
c  calculate index of mass storage
c
      m1 = (n1-1)*ndof
      m2 = (n2-1)*ndof
      m3 = (n3-1)*ndof
      m4 = (n4-1)*ndof
      m5 = (n5-1)*ndof
      m6 = (n6-1)*ndof
      m7 = (n7-1)*ndof
      m8 = (n8-1)*ndof
c
c  assemble to global mass matrix --- xmass(8)
c
      xmass(m1+1) = xmass(m1+1) + totmass/8.d0
      xmass(m1+2) = xmass(m1+2) + totmass/8.d0
      xmass(m1+3) = xmass(m1+3) + totmass/8.d0
      xmass(m2+1) = xmass(m2+1) + totmass/8.d0
      xmass(m2+2) = xmass(m2+2) + totmass/8.d0
```

```
      xmass(m2+3) = xmass(m2+3) + totmass/8.d0
      xmass(m3+1) = xmass(m3+1) + totmass/8.d0
      xmass(m3+2) = xmass(m3+2) + totmass/8.d0
      xmass(m3+3) = xmass(m3+3) + totmass/8.d0
      xmass(m4+1) = xmass(m4+1) + totmass/8.d0
      xmass(m4+2) = xmass(m4+2) + totmass/8.d0
      xmass(m4+3) = xmass(m4+3) + totmass/8.d0
      xmass(m5+1) = xmass(m5+1) + totmass/8.d0
      xmass(m5+2) = xmass(m5+2) + totmass/8.d0
      xmass(m5+3) = xmass(m5+3) + totmass/8.d0
      xmass(m6+1) = xmass(m6+1) + totmass/8.d0
      xmass(m6+2) = xmass(m6+2) + totmass/8.d0
      xmass(m6+3) = xmass(m6+3) + totmass/8.d0
      xmass(m7+1) = xmass(m7+1) + totmass/8.d0
      xmass(m7+2) = xmass(m7+2) + totmass/8.d0
      xmass(m7+3) = xmass(m7+3) + totmass/8.d0
      xmass(m8+1) = xmass(m8+1) + totmass/8.d0
      xmass(m8+2) = xmass(m8+2) + totmass/8.d0
      xmass(m8+3) = xmass(m8+3) + totmass/8.d0
   20 continue
      return
      end
c
*****************************************************************************
*                                                                           *
*  compute elastic constitutive coefficients under  3D condition            *
*                                                                           *
*                                                                           *
*****************************************************************************
c
      subroutine elastd3d (eyng,poisson,ed)
c
      implicit real*8 (a-h,o-z)
      dimension ed(6,6)
c
      do 50 i=1,6
      do 60 j=1,6
       ed(i,j) = 0.d0
   60 continue
   50 continue
c
      G  = eyng/(2.d0*(1.d0+poisson))
      dla  = poisson*eyng/((1.d0+poisson)*(1.d0-2.d0*poisson))
c
      ed(1,1) = 2.d0*G + dla
      ed(2,2) = ed(1,1)
      ed(3,3) = ed(1,1)
      ed(4,4) = G
      ed(5,5) = ed(4,4)
      ed(6,6) = ed(4,4)
      ed(1,2) = dla
      ed(2,1) = ed(1,2)
      ed(3,1) = ed(1,2)
      ed(3,2) = ed(1,2)
      ed(1,3) = ed(1,2)
      ed(2,3) = ed(1,2)
c
      return
      end
c
```

```
************************************************************************
*                                                                      *
*  Solve Eq. of Motion by Displacement-based Central Difference Method *
*                                                                      *
************************************************************************
c
      subroutine esolv (time,nel,nnd,ndof,iacc,numout,iforce,xmass,
     +                  force,pint,rifix,d,v,a,xc,yc,zc,rnode,e,rkout,
     +                  maxstp,dn,dp,inital,meq,sigmaP,sigmaN,epslonP,
     +                  epslonN,PLalphaP,PLalphaN,PLrP,PLrN,elplas,
     +                  elpout,proutv,s1,s2,s3,s4,s5,s6,s7)
c
      implicit real*8 (a-h,o-z)
      dimension xmass(1),force(1),pint(1),rnode(12,1),rkout(3,1)
      dimension d(1),v(1),a(1),xc(1),yc(1),zc(1),e(11,1),rifix(1)
      dimension dn(1),dp(1),ag(3),elplas(1),elpout(1)
      dimension sigmaP(1),sigmaN(1),epslonP(1),epslonN(1),proutv(1)
      dimension PLalphaP(1),PLalphaN(1),PLrP(1),PLrN(1)
      dimension s1(1),s2(1),s3(1),s4(1),s5(1),s6(1),s7(1)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
      common /box 4/ npnts,numif,nnaf,ndisi,nveli,
     +               kfpnts(6),jnode(30),jdir(30),jaf(30),
     +               ndnod(10),kdis(10),nvnod(10),kvel(10)
      common /box 4a/g,disi(10),veli(10),f(10),omega(10),
     +               ff(500,6),ta(500,6),tf(500,6),aa(500,6)
c
c  define initial condition that time is within the range
c     -- see subroutine finter in detail
      noyes = 0
c
      do 100 i=1,meq
         d(i) = 0.d0
         v(i) = 0.d0
         a(i) = 0.d0
  100 continue
c
      if (inital .eq. 0) go to 50
      if (inital .eq. 2) go to 60
c
c  set index no. of d,v, and put initial value into them
c
      do 65 i=1,ndisi
      ind = (ndnod(i)-1)*ndof+kdis(i)
      d(ind) = disi(i)
   65 continue
c
      if (inital .ne. 3) go to 50
c
   60 continue
      do 70 i=1,nveli
      inv = (nvnod(i)-1)*ndof+kvel(i)
      v(inv) = veli(i)
   70 continue

c
c  compute displacement (nstep=0) before first step displ.(nstep=1)
c  by Central Difference Method
c
   50 continue
```

```
      do 75 i=1,meq
       dn(i) = d(i)-delta*v(i)+0.5*delta*delta*a(i)
   75 continue
c
      nstep = 0
      nskip = 0
c
  160 continue
c
      do 120 i=1,meq
       force(i) = 0.d0
  120 continue
c
c impose gravity load
c
      do 140 i=2,meq,3
       force(i) = -1.d0*gravity*xmass(i)
  140 continue
c
c impose impact force
c  set index no. of nodal d.o.f. applied by impact force
c  compute external force at that time step by linear interpolation
c
      if (iforce .eq. 0) go to 167
       do 165 n=1,nnaf
      imn = (jnode(n)-1)*ndof+jdir(n)
      if (iforce .ne. 1) go to 163
      ma = jaf(n)
      mb = kfpnts(ma)
      call finter (ff(1,ma),tf(1,ma),mb,time,pht,noyes)
      force(imn) = pht + force(imn)
      go to 165
c
  163 if(iforce.eq.2)force(imn)=f(n)*dsin(omega(n)*time)+force(imn)
      if(iforce.eq.3)force(imn)=f(n)*dcos(omega(n)*time)+force(imn)
  165 continue
  167 continue

c
c impose ground acceleration
c
      if (iacc .eq. 0) go to 25
c
      do 5 i=1,3
       ag(i) = 0.d0
    5 continue
c
c  compute acceleration at that time step by linear interpolation
c
      if (iacc .lt. 4) go to 13
      if (iacc .eq. 4) go to 12
c
      do 11 j=1,3
       call finter (aa(1,j),ta(1,j),npnts,time,pht,noyes)
       ag(j) = pht
   11 continue
      go to 14
c
   12 continue
      call finter (aa(1,1),ta(1,1),npnts,time,pht,noyes)
```

```
      ag(1) = pht
      call finter (aa(1,2),ta(1,2),npnts,time,pht,noyes)
      ag(3) = pht
      go to 14
c
   13 call finter (aa(1,1),ta(1,1),npnts,time,pht,noyes)
      ag(iacc) = pht
c
c  compute external force by using  d'Alembert principle to account for
c  inertia force
c
   14 continue
      do 20 i=1,nnd
        ii = (i-1)*ndof
      do 21 j=ii+1,ii+3
        force(j) = force(j)-xmass(j)*ag(j-ii)*g
   21 continue
   20 continue
   25 continue
c
c initialized index for average stress output
c
c
      do 998 ig = 1,numout
      proutv(ig) = 0.d0
  998 continue
c
      do 999 io = 1,nnd
      s1(io) = 0.0d0
      s2(io) = 0.0d0
      s3(io) = 0.0d0
      s4(io) = 0.0d0
      s5(io) = 0.0d0
      s6(io) = 0.0d0
      s7(io) = 0.0d0
  999 continue
c
      do 991 ki = 1,8*nel
      elplas(ki) = 0.0d0
      elpout(ki) = 0.0d0
  991 continue
c
c  compute element internal nodal forces(pint)
c
      do 22 ix=1,meq
       pint(ix) = 0.d0
   22 continue
c
      do 23 i = 1,nel
       neo = int(rnode(12,i))
       if (neo.lt.0  .or. neo.eq.11) goto 23
      call fintiso8 (i,ndof,xc,yc,zc,rnode,e,d,pint,sigmaP,
     +               sigmaN,epslonP,epslonN,PLalphaP,PLalphaN,
     +               PLrP,PLrN,elplas,s1,s2,s3,s4,s5,s6,s7,time)
   23 continue
c
      do 360 i=1,nel
       n = (i-1)*8
       do 360 k=1,8
      do 370 j=1,6
```

```
        m = (n+k-1)*6+j
        sigmaP(m) = sigmaN(m)
        epslonP(m) = epslonN(m)
        PLalphaP(m) = PLalphaN(m)
  370 continue
        PLrP(n+k) = PLrN(n+k)
  360 continue
c
c  compute displacement of next time step        --- dp(j)
c  and velocity & acceleration of this time step --- v(j) & a(j)
c  by Displacement-based Central Difference Method
c
      do 170 i=1,nnd
       m = (i-1)*ndof
      do 171 j=m+1,m+3
       if (int(rifix(j)) .eq. 1) go to 171
        if (xmass(j) .le. 0.d0) go to 171
        a1 = 2.d0+alpha*delta
        a2 = 2.d0-alpha*delta
        c1 = (2.d0/a1)*delta*delta
        c2 = 4.d0/a1
        c3 = a2/a1
        c4 = (force(j)-pint(j))/xmass(j)
       dp(j) = c1*c4+c2*d(j)-c3*dn(j)
        v(j) = (dp(j)-dn(j))/(2.d0*delta)
        a(j) = (dp(j)-2.d0*d(j)+dn(j))/(delta*delta)
  171 continue
  170 continue

c
c
c record plastic element number
c
      if (nskip .eq. iprob) then
      write(6,*) 'nstep=',nstep
      ij = 0
      do 556 i=1,8*nel
      if (elplas(i) .lt. 1.0) go to 556
      ij = ij + 1
      elpout(ij) = elplas(i)
  556 continue
      write (6,2021)
 2021 format(/,'  Plastic element no [element no.Gauss point no] =')
      write (6,2031) (elpout(i),i=1,ij)
 2031 format(8(3x,f5.1))
      if (ij .eq. 0) write(6,2032)
 2032 format('      NONE')
      endif
c
c  print out response results requested every "iprob" step
c
      if (nskip .ne. iprob) go to 195
      if (numout .ne. 0) then
      call prout (numout,rkout,d,v,a,s1,s2,s3,s4,s5,s6,s7,ndof,nstep,
     +            time,proutv)
      nskip = 0
      endif
  195 continue
      if (nstep .ge. maxstp) go to 225
c
```

```
c  check whether the increment of each nodal displacement is less than
c  the tolerance limit, i.e. TOLER.
c
      if(nstep .lt. 200) go to 198
      ddmax = 0.0d0
      do 197 i=1,meq
         yn = dabs(dp(i)-d(i))
  197 if (ddmax .lt. yn) ddmax = yn
      if(ddmax .lt. toler) then
       write (*,1100) ddmax
 1100 format (/,' STOP -- displ.increment < tolerance limit',1pd9.3,/)
      go to 225
      endif
  198 continue
c
      do 200 i=1,meq
       dn(i) = d(i)
       d(i)  = dp(i)
  200 continue
c
      nskip = nskip + 1
      nstep = nstep + 1
      time  = time + delta
      go to 160
c
  225 continue
c
c record plastic element numberfor last time step
c
      write(6,*) 'nstep=',nstep
      ij = 0
      do 555 i=1,8*nel
      if (elplas(i) .lt. 1.0) go to 555
      ij = ij + 1
      elpout(ij) = elplas(i)
  555 continue
      write (6,2001)
 2001 format(/,'  Plastic element no =>[Element no.Gauss point no] =')
      write (6,2011) (elpout(i),i=1,ij)
 2011 format(8(3x,f5.1))
      if (ij .eq. 0) write(6,2022)
 2022 format('      NONE')
      print *,'complete!'
c
      return
      end
c
*************************************************************************
*                                                                       *
*  calculate each time step's external force and acceleration by linear*
*  interpolation ---- because of mismatch between the time interval of *
*  force & acc. input data, and the time increment                       *
*                                                                       *
*************************************************************************
c
      subroutine finter (pp,vv,kc,t,pht,noyes)
c
      implicit real*8 (a-h,o-z)
      dimension pp(kc),vv(kc)
c
```

```
      if (t.lt.vv(1) .or. t.gt.vv(kc)) then
       if (noyes .eq. 1) go to  900
       write(6,*)' '
       write(6,*)' WARNING -- time out of the range of time-force pairs'
       write(6,*)'             ZERO FORCE is given at that time.'
       write(6,*)'             the out-of-the-range time is ',t
       write(6,*)' '
       print *, ' '
       print *, ' WARNING -- time out of the range of time-force pairs'
       print *, '             ZERO FORCE is given at that time.'
       print *, '             the out-of-the-range time is ',t
       print *, ' '
       noyes = 1
  900    pht = 0.0
       go to 140
      endif
c
      do 100 l=2,kc
       if ( t .le. vv(l)) go to 120
  100 continue
c
  120 pht = pp(l-1)+(t-vv(l-1))*(pp(l)-pp(l-1))/(vv(l)-vv(l-1))
c
  140 return
c
      end
c
*************************************************************************
*                                                                       *
*  Compute Element Internal Nodal Forces (pint)   by  ---               *
*  8-node bicubic isoparametric element (8-node solid element)          *
*                                                                       *
*************************************************************************
c
      subroutine fintiso8 (i,ndof,xc,yc,zc,rnode,e,d,pint,sigmaP,
     +        sigmaN,epslonP,epslonN,PLalphaP,PLalphaN,PLrP,PLrN,elplas,
     +        s1,s2,s3,s4,s5,s6,s7,time)
c
      implicit real*8 (a-h,o-z)
      dimension xc(1),yc(1),zc(1),rnode(12,1),e(11,1),d(1),pint(1)
      dimension disl(24),f(24),b(6,24),te(6,6)
      dimension epsx(8),epsy(8),epsz(8),epsxy(8),epsyz(8),epszx(8)
      dimension ed(6,6),sigma(6),sig(8,6)
      dimension Depslon(6),PLalpha(6),elplas(1)
      dimension sigmaP(1),sigmaN(1),epslonP(1),epslonN(1)
      dimension PLalphaP(1),PLalphaN(1),PLrP(1),PLrN(1)
      dimension s(8),t(8),r(8),w(8)
      dimension s1(1),s2(1),s3(1),s4(1),s5(1),s6(1),s7(1)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
      data phi/3.141592653589793d0/
c
c  matching local to global nodal number for one element
c
      n1 = int(rnode(1,i))
      n2 = int(rnode(2,i))
      n3 = int(rnode(3,i))
      n4 = int(rnode(4,i))
      n5 = int(rnode(5,i))
      n6 = int(rnode(6,i))
```

```
      n7 = int(rnode(7,i))
      n8 = int(rnode(8,i))
c
c  matching local to global coordinates for one element
c
      x1 = xc(n1)
      x2 = xc(n2)
      x3 = xc(n3)
      x4 = xc(n4)
      x5 = xc(n5)
      x6 = xc(n6)
      x7 = xc(n7)
      x8 = xc(n8)
c
      y1 = yc(n1)
      y2 = yc(n2)
      y3 = yc(n3)
      y4 = yc(n4)
      y5 = yc(n5)
      y6 = yc(n6)
      y7 = yc(n7)
      y8 = yc(n8)
c
      z1 = zc(n1)
      z2 = zc(n2)
      z3 = zc(n3)
      z4 = zc(n4)
      z5 = zc(n5)
      z6 = zc(n6)
      z7 = zc(n7)
      z8 = zc(n8)
c
c  node(12) = 1 ---  element condition ( read in sub"readata" )
c  if node(12) = neo = 11  ---> element has a negative Jacobian
c
      neo = int(rnode(12,i))
c
      if (neo.eq.11) go to 13
c
c  set index no. of nodes
c
      m1 = (n1-1)*ndof
      m2 = (n2-1)*ndof
      m3 = (n3-1)*ndof
      m4 = (n4-1)*ndof
      m5 = (n5-1)*ndof
      m6 = (n6-1)*ndof
      m7 = (n7-1)*ndof
      m8 = (n8-1)*ndof
c
c  set node displacement
c
      d1x = d(m1+1)
      d1y = d(m1+2)
      d1z = d(m1+3)
c
      d2x = d(m2+1)
      d2y = d(m2+2)
      d2z = d(m2+3)
c
```

```
      d3x = d(m3+1)
      d3y = d(m3+2)
      d3z = d(m3+3)
c
      d4x = d(m4+1)
      d4y = d(m4+2)
      d4z = d(m4+3)
c
      d5x = d(m5+1)
      d5y = d(m5+2)
      d5z = d(m5+3)
c
      d6x = d(m6+1)
      d6y = d(m6+2)
      d6z = d(m6+3)
c
      d7x = d(m7+1)
      d7y = d(m7+2)
      d7z = d(m7+3)
c
      d8x = d(m8+1)
      d8y = d(m8+2)
      d8z = d(m8+3)
c
c deformed coordinates
c
      x1d = x1 + d1x
      y1d = y1 + d1y
      z1d = z1 + d1z
      x2d = x2 + d2x
      y2d = y2 + d2y
      z2d = z2 + d2z
      x3d = x3 + d3x
      y3d = y3 + d3y
      z3d = z3 + d3z
      x4d = x4 + d4x
      y4d = y4 + d4y
      z4d = z4 + d4z
      x5d = x5 + d5x
      y5d = y5 + d5y
      z5d = z5 + d5z
      x6d = x6 + d6x
      y6d = y6 + d6y
      z6d = z6 + d6z
      x7d = x7 + d7x
      y7d = y7 + d7y
      z7d = z7 + d7z
      x8d = x8 + d8x
      y8d = y8 + d8y
      z8d = z8 + d8z
c
c  find rotational angles of element,
c  based on the vector from centroid to each node projected into
c  three planes: xy, xz, and zy
c  the coordinates of the centroid of element = (xcd,ycd,zcd)
      xcd1 = (x1+x2+x3+x4+x5+x6+x7+x8)/8.d0
      ycd1 = (y1+y2+y3+y4+y5+y6+y7+y8)/8.d0
      zcd1 = (z1+z2+z3+z4+z5+z6+z7+z8)/8.d0
c
      xcd2 = (x1d+x2d+x3d+x4d+x5d+x6d+x7d+x8d)/8.d0
```

```
      ycd2 = (y1d+y2d+y3d+y4d+y5d+y6d+y7d+y8d)/8.d0
      zcd2 = (z1d+z2d+z3d+z4d+z5d+z6d+z7d+z8d)/8.d0
c
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x1,y1,z1,x1d,y1d,z1d,ttdx1,ttdy1,ttdz1)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x2,y2,z2,x2d,y2d,z2d,ttdx2,ttdy2,ttdz2)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x3,y3,z3,x3d,y3d,z3d,ttdx3,ttdy3,ttdz3)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x4,y4,z4,x4d,y4d,z4d,ttdx4,ttdy4,ttdz4)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x5,y5,z5,x5d,y5d,z5d,ttdx5,ttdy5,ttdz5)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x6,y6,z6,x6d,y6d,z6d,ttdx6,ttdy6,ttdz6)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x7,y7,z7,x7d,y7d,z7d,ttdx7,ttdy7,ttdz7)
      call thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                x8,y8,z8,x8d,y8d,z8d,ttdx8,ttdy8,ttdz8)
c
      tx = (ttdx1+ttdx2+ttdx3+ttdx4+ttdx5+ttdx6+ttdx7+ttdx8)/8.d0
      ty = (ttdy1+ttdy2+ttdy3+ttdy4+ttdy5+ttdy6+ttdy7+ttdy8)/8.d0
      tz = (ttdz1+ttdz2+ttdz3+ttdz4+ttdz5+ttdz6+ttdz7+ttdz8)/8.d0
c
      if (dabs(tx) .le. 1.0d-05) tx = 0.0d0
      if (dabs(ty) .le. 1.0d-05) ty = 0.0d0
      if (dabs(tz) .le. 1.0d-05) tz = 0.0d0
c
c  create transformation matrix --- global to local
c
      rl1 =  dcos(tz)*dcos(ty) - dsin(tx)*dsin(ty)*dsin(tz)
      rm1 =  dcos(ty)*dsin(tz) + dcos(tz)*dsin(ty)*dsin(tx)
      rn1 = -dcos(tx)*dsin(ty)
      rl2 = -dcos(tx)*dsin(tz)
      rm2 =  dcos(tz)*dcos(tx)
      rn2 =  dsin(tx)
      rl3 =  dsin(ty)*dcos(tz) + dcos(ty)*dsin(tx)*dsin(tz)
      rm3 =  dsin(ty)*dsin(tz) - dcos(ty)*dcos(tz)*dsin(tx)
      rn3 =  dcos(ty)*dcos(tx)
c
c  compute local node coord. and displ. ( rotation from global to local
)
c
      xl1 = rl1*x1 + rm1*y1 + rn1*z1
      yl1 = rl2*x1 + rm2*y1 + rn2*z1
      zl1 = rl3*x1 + rm3*y1 + rn3*z1
      xl2 = rl1*x2 + rm1*y2 + rn1*z2
      yl2 = rl2*x2 + rm2*y2 + rn2*z2
      zl2 = rl3*x2 + rm3*y2 + rn3*z2
      xl3 = rl1*x3 + rm1*y3 + rn1*z3
      yl3 = rl2*x3 + rm2*y3 + rn2*z3
      zl3 = rl3*x3 + rm3*y3 + rn3*z3
      xl4 = rl1*x4 + rm1*y4 + rn1*z4
      yl4 = rl2*x4 + rm2*y4 + rn2*z4
      zl4 = rl3*x4 + rm3*y4 + rn3*z4
      xl5 = rl1*x5 + rm1*y5 + rn1*z5
      yl5 = rl2*x5 + rm2*y5 + rn2*z5
      zl5 = rl3*x5 + rm3*y5 + rn3*z5
      xl6 = rl1*x6 + rm1*y6 + rn1*z6
      yl6 = rl2*x6 + rm2*y6 + rn2*z6
```

```
      zl6 = rl3*x6 + rm3*y6 + rn3*z6
      xl7 = rl1*x7 + rm1*y7 + rn1*z7
      yl7 = rl2*x7 + rm2*y7 + rn2*z7
      zl7 = rl3*x7 + rm3*y7 + rn3*z7
      xl8 = rl1*x8 + rm1*y8 + rn1*z8
      yl8 = rl2*x8 + rm2*y8 + rn2*z8
      zl8 = rl3*x8 + rm3*y8 + rn3*z8
c
      disl(1)  =  rl1*d1x + rm1*d1y + rn1*d1z
      disl(2)  =  rl2*d1x + rm2*d1y + rn2*d1z
      disl(3)  =  rl3*d1x + rm3*d1y + rn3*d1z
c
      disl(4)  =  rl1*d2x + rm1*d2y + rn1*d2z
      disl(5)  =  rl2*d2x + rm2*d2y + rn2*d2z
      disl(6)  =  rl3*d2x + rm3*d2y + rn3*d2z
c
      disl(7)  =  rl1*d3x + rm1*d3y + rn1*d3z
      disl(8)  =  rl2*d3x + rm2*d3y + rn2*d3z
      disl(9)  =  rl3*d3x + rm3*d3y + rn3*d3z
c
      disl(10) =  rl1*d4x + rm1*d4y + rn1*d4z
      disl(11) =  rl2*d4x + rm2*d4y + rn2*d4z
      disl(12) =  rl3*d4x + rm3*d4y + rn3*d4z
c
      disl(13) =  rl1*d5x + rm1*d5y + rn1*d5z
      disl(14) =  rl2*d5x + rm2*d5y + rn2*d5z
      disl(15) =  rl3*d5x + rm3*d5y + rn3*d5z
c
      disl(16) =  rl1*d6x + rm1*d6y + rn1*d6z
      disl(17) =  rl2*d6x + rm2*d6y + rn2*d6z
      disl(18) =  rl3*d6x + rm3*d6y + rn3*d6z
c
      disl(19) =  rl1*d7x + rm1*d7y + rn1*d7z
      disl(20) =  rl2*d7x + rm2*d7y + rn2*d7z
      disl(21) =  rl3*d7x + rm3*d7y + rn3*d7z
c
      disl(22) =  rl1*d8x + rm1*d8y + rn1*d8z
      disl(23) =  rl2*d8x + rm2*d8y + rn2*d8z
      disl(24) =  rl3*d8x + rm3*d8y + rn3*d8z
c
c  Initialized the stresses and internal forces at each Gauss's point
c
      do 100 j=1,8
       epsx(j)  = 0.d0
       epsy(j)  = 0.d0
       epsz(j)  = 0.d0
       epsxy(j) = 0.d0
       epsyz(j) = 0.d0
       epszx(j) = 0.d0
  100 continue
c
      do 109 ijj = 1,24
       f(ijj) = 0.d0
  109 continue
c
c  coeff. of 8 point Gauss Quadrature: s(i),t(i),r(i),w(i);i=1,8
c
      const = 1.d0/dsqrt(3.0d0)
      s(1) = -const
      t(1) = -const
```

```
      r(1) = const
c
      s(2) = const
      t(2) = -const
      r(2) = const
c
      s(3) = const
      t(3) = const
      r(3) = const
c
      s(4) = -const
      t(4) = const
      r(4) = const
c
      s(5) = -const
      t(5) = -const
      r(5) = -const
c
      s(6) = const
      t(6) = -const
      r(6) = -const
c
      s(7) = const
      t(7) = const
      r(7) = -const
c
      s(8) = -const
      t(8) = const
      r(8) = -const
c
      do 199 iii = 1,8
      w(iii) = 1.d0
  199 continue
c
c  set shape function Ni => k = gauss'spoint no.
c
      do 201 k=1,8
       shpf1 = (1.d0-s(k))*(1.d0-t(k))*(1.d0+r(k))/8.d0
       shpf2 = (1.d0+s(k))*(1.d0-t(k))*(1.d0+r(k))/8.d0
       shpf3 = (1.d0+s(k))*(1.d0+t(k))*(1.d0+r(k))/8.d0
       shpf4 = (1.d0-s(k))*(1.d0+t(k))*(1.d0+r(k))/8.d0
       shpf5 = (1.d0-s(k))*(1.d0-t(k))*(1.d0-r(k))/8.d0
       shpf6 = (1.d0+s(k))*(1.d0-t(k))*(1.d0-r(k))/8.d0
       shpf7 = (1.d0+s(k))*(1.d0+t(k))*(1.d0-r(k))/8.d0
       shpf8 = (1.d0-s(k))*(1.d0+t(k))*(1.d0-r(k))/8.d0
c
c  derivatives of shape function w.r.t. s,t,r
c
       s1s = -(1.d0-t(k))*(1.d0+r(k)) / 8.d0
       s2s =  (1.d0-t(k))*(1.d0+r(k)) / 8.d0
       s3s =  (1.d0+t(k))*(1.d0+r(k)) / 8.d0
       s4s = -(1.d0+t(k))*(1.d0+r(k)) / 8.d0
       s5s = -(1.d0-t(k))*(1.d0-r(k)) / 8.d0
       s6s =  (1.d0-t(k))*(1.d0-r(k)) / 8.d0
       s7s =  (1.d0+t(k))*(1.d0-r(k)) / 8.d0
       s8s = -(1.d0+t(k))*(1.d0-r(k)) / 8.d0
c
       s1t = -(1.d0-s(k))*(1.d0+r(k)) / 8.d0
       s2t = -(1.d0+s(k))*(1.d0+r(k)) / 8.d0
       s3t =  (1.d0+s(k))*(1.d0+r(k)) / 8.d0
```

```
      s4t =  (1.d0-s(k))*(1.d0+r(k)) / 8.d0
      s5t = -(1.d0-s(k))*(1.d0-r(k)) / 8.d0
      s6t = -(1.d0+s(k))*(1.d0-r(k)) / 8.d0
      s7t =  (1.d0+s(k))*(1.d0-r(k)) / 8.d0
      s8t =  (1.d0-s(k))*(1.d0-r(k)) / 8.d0
c
      s1r =  (1.d0-s(k))*(1.d0-t(k)) / 8.d0
      s2r =  (1.d0+s(k))*(1.d0-t(k)) / 8.d0
      s3r =  (1.d0+s(k))*(1.d0+t(k)) / 8.d0
      s4r =  (1.d0-s(k))*(1.d0+t(k)) / 8.d0
      s5r = -(1.d0-s(k))*(1.d0-t(k)) / 8.d0
      s6r = -(1.d0+s(k))*(1.d0-t(k)) / 8.d0
      s7r = -(1.d0+s(k))*(1.d0+t(k)) / 8.d0
      s8r = -(1.d0-s(k))*(1.d0+t(k)) / 8.d0
c
c  Jacobian Matrix Element(Jik)
c
      xJ11 = s1s*xl1+s2s*xl2+s3s*xl3+s4s*xl4+
     +       s5s*xl5+s6s*xl6+s7s*xl7+s8s*xl8
      xJ12 = s1s*yl1+s2s*yl2+s3s*yl3+s4s*yl4+
     +       s5s*yl5+s6s*yl6+s7s*yl7+s8s*yl8
      xJ13 = s1s*zl1+s2s*zl2+s3s*zl3+s4s*zl4+
     +       s5s*zl5+s6s*zl6+s7s*zl7+s8s*zl8
      xJ21 = s1t*xl1+s2t*xl2+s3t*xl3+s4t*xl4+
     +       s5t*xl5+s6t*xl6+s7t*xl7+s8t*xl8
      xJ22 = s1t*yl1+s2t*yl2+s3t*yl3+s4t*yl4+
     +       s5t*yl5+s6t*yl6+s7t*yl7+s8t*yl8
      xJ23 = s1t*zl1+s2t*zl2+s3t*zl3+s4t*zl4+
     +       s5t*zl5+s6t*zl6+s7t*zl7+s8t*zl8
      xJ31 = s1r*xl1+s2r*xl2+s3r*xl3+s4r*xl4+
     +       s5r*xl5+s6r*xl6+s7r*xl7+s8r*xl8
      xJ32 = s1r*yl1+s2r*yl2+s3r*yl3+s4r*yl4+
     +       s5r*yl5+s6r*yl6+s7r*yl7+s8r*yl8
      xJ33 = s1r*zl1+s2r*zl2+s3r*zl3+s4r*zl4+
     +       s5r*zl5+s6r*zl6+s7r*zl7+s8r*zl8

c
c  Jacobian = det|J|
c
      down     = (xJ11*xJ22*xJ33 + xJ12*xJ23*xJ31 + xJ13*xJ21*xJ32)
      up       = (xJ31*xJ22*xJ13 + xJ32*xJ23*xJ11 + xJ33*xJ21*xJ12)
      detJ     = down - up
c  check negative jacobian
      if (detJ .le. 0.0) then
      if (k .eq. 1) then
      print *,  '--- Negative Jacobian, Node No. = ', n1 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n1 ,' J= ', detJ
       else if (k .eq. 2) then
      print *,  '--- Negative Jacobian, Node No. = ', n2 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n2 ,' J= ', detJ
       else if (k .eq. 3) then
      print *,  '--- Negative Jacobian, Node No. = ', n3 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n3 ,' J= ', detJ
       else if (k .eq. 4) then
       print *,  '--- Negative Jacobian, Node No. = ', n4 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n4 ,' J= ', detJ
       else if (k .eq. 5) then
      print *,  '--- Negative Jacobian, Node No. = ', n5 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n5 ,' J= ', detJ
      else if (k .eq. 6) then
```

```
      print *,  '--- Negative Jacobian, Node No. = ', n6 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n6 ,' J= ', detJ
       else if (k .eq. 7) then
      print *,  '--- Negative Jacobian, Node No. = ', n7 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n7 ,' J= ', detJ
       else if (k .eq. 8) then
      print *,  '--- Negative Jacobian, Node No. = ', n8 ,' J= ', detJ
      write(6,*)'--- Negative Jacobian, Node No. = ', n8 ,' J= ', detJ
       end if
       rnode(12,i) = 11.d0
      go to 13
      endif
c
c  form inverse jacobian
c
      a11 =  (xJ22*xJ33 - xJ32*xJ23)/detJ
      a12 = -(xJ12*xJ33 - xJ32*xJ13)/detJ
      a13 =  (xJ12*xJ23 - xJ22*xJ13)/detJ
      a21 = -(xJ21*xJ33 - xJ31*xJ23)/detJ
      a22 =  (xJ11*xJ33 - xJ31*xJ13)/detJ
      a23 = -(xJ11*xJ23 - xJ21*xJ13)/detJ
      a31 =  (xJ21*xJ32 - xJ31*xJ22)/detJ
      a32 = -(xJ11*xJ23 - xJ21*xJ13)/detJ
      a33 =  (xJ11*xJ22 - xJ21*xJ12)/detJ
c
c  form B matrix
c
      b1x = a11*s1s+a12*s1t+a13*s1r
      b2x = a11*s2s+a12*s2t+a13*s2r
      b3x = a11*s3s+a12*s3t+a13*s3r
      b4x = a11*s4s+a12*s4t+a13*s4r
      b5x = a11*s5s+a12*s5t+a13*s5r
      b6x = a11*s6s+a12*s6t+a13*s6r
      b7x = a11*s7s+a12*s7t+a13*s7r
      b8x = a11*s8s+a12*s8t+a13*s8r
c
      b1y = a21*s1s+a22*s1t+a23*s1r
      b2y = a21*s2s+a22*s2t+a23*s2r
      b3y = a21*s3s+a22*s3t+a23*s3r
      b4y = a21*s4s+a22*s4t+a23*s4r
      b5y = a21*s5s+a22*s5t+a23*s5r
      b6y = a21*s6s+a22*s6t+a23*s6r
      b7y = a21*s7s+a22*s7t+a23*s7r
      b8y = a21*s8s+a22*s8t+a23*s8r
c
      b1z = a31*s1s+a32*s1t+a33*s1r
      b2z = a31*s2s+a32*s2t+a33*s2r
      b3z = a31*s3s+a32*s3t+a33*s3r
      b4z = a31*s4s+a32*s4t+a33*s4r
      b5z = a31*s5s+a32*s5t+a33*s5r
      b6z = a31*s6s+a32*s6t+a33*s6r
      b7z = a31*s7s+a32*s7t+a33*s7r
      b8z = a31*s8s+a32*s8t+a33*s8r
c
      do 21 ie=1,6
      do 22 le=1,24
       b(ie,le) = 0.d0
   22 continue
   21 continue
      b(1,1)  = b1x
```

```
b(1,4)  = b2x
b(1,7)  = b3x
b(1,10) = b4x
b(1,13) = b5x
b(1,16) = b6x
b(1,19) = b7x
b(1,22) = b8x
b(2,2)  = b1y
b(2,5)  = b2y
b(2,8)  = b3y
b(2,11) = b4y
b(2,14) = b5y
b(2,17) = b6y
b(2,20) = b7y
b(2,23) = b8y
b(3,3)  = b1z
b(3,6)  = b2z
b(3,9)  = b3z
b(3,12) = b4z
b(3,15) = b5z
b(3,18) = b6z
b(3,21) = b7z
b(3,24) = b8z
b(4,1)  = b1y
b(4,2)  = b1x
b(4,4)  = b2y
b(4,5)  = b2x
b(4,7)  = b3y
b(4,8)  = b3x
b(4,10) = b4y
b(4,11) = b4x
b(4,13) = b5y
b(4,14) = b5x
b(4,16) = b6y
b(4,17) = b6x
b(4,19) = b7y
b(4,20) = b7x
b(4,22) = b8y
b(4,23) = b8x
b(5,2)  = b1z
b(5,3)  = b1y
b(5,5)  = b2z
b(5,6)  = b2y
b(5,8)  = b3z
b(5,9)  = b3y
b(5,11) = b4z
b(5,12) = b4y
b(5,14) = b5z
b(5,15) = b5y
b(5,17) = b6z
b(5,18) = b6y
b(5,20) = b7z
b(5,21) = b7y
b(5,23) = b8z
b(5,24) = b8y
b(6,1)  = b1z
b(6,3)  = b1x
b(6,4)  = b2z
b(6,6)  = b2x
b(6,7)  = b3z
```

```
      b(6,9)  = b3x
      b(6,10) = b4z
      b(6,12) = b4x
      b(6,13) = b5z
      b(6,15) = b5x
      b(6,16) = b6z
      b(6,18) = b6x
      b(6,19) = b7z
      b(6,21) = b7x
      b(6,22) = b8z
      b(6,24) = b8x
c
c  compute local element strains of each Gauss's point
c
      do 300 j=1,24
       epsx(k)  = epsx(k)  + b(1,j)*disl(j)
       epsy(k)  = epsy(k)  + b(2,j)*disl(j)
       epsz(k)  = epsz(k)  + b(3,j)*disl(j)
       epsxy(k) = epsxy(k) + b(4,j)*disl(j)
       epsyz(k) = epsyz(k) + b(5,j)*disl(j)
       epszx(k) = epszx(k) + b(6,j)*disl(j)
  300 continue
c
c assign value of new epslon (epslonN(..)
c
      m = ((i-1)*8+k-1)*6
      epslonN(m+1) = epsx(k)
      epslonN(m+2) = epsy(k)
      epslonN(m+3) = epsz(k)
      epslonN(m+4) = epsxy(k)
      epslonN(m+5) = epsyz(k)
      epslonN(m+6) = epszx(k)
c
c calculate delta epslon
c
      do 352 in = 1,6
       Depslon(in) = epslonN(m+in) - epslonP(m+in)
  352 continue
c
c  material properties for element
c
      mtyp    = int(rnode(9,i))
      mtyp2   = int(e(1,mtyp))
      eyng    = e(3,mtyp)
      if (eyng .le. 0.0d0) goto 13
      poisson = e(4,mtyp)
      ft      = e(5,mtyp)
      coh     = e(6,mtyp)
      phiangle = e(7,mtyp)*phi/180.d0
      kickPL   = int(e(8,mtyp))
      eyngt    = e(9,mtyp)
      beta     = e(10,mtyp)
      tau      = e(11,1)
c
c  call constitutive coefficient from subroutine.
c
      call elastd3d (eyng,poisson,ed)
c
c calculate trial stress(sigma = stresses at gauss pts)
c
```

```
      ny = (i-1)*8 + k
      PLr = PLrP(ny)
      do 353 m=1,6
       ny = ((i-1)*8+k-1)*6+m
       sigma(m) = sigmaP(ny)
       PLalpha(m) = PLalphaP(ny)
  353 continue
c
      if (kickPL .eq. 1) then
      do 402 m = 1,6
      do 502 n = 1,6
      ed(m,n) = exp(-time/tau)*ed(m,n)
  502 continue
  402 continue
      end if
c
      do 400 m = 1,6
      do 500 n = 1,6
         sigma(m) = sigma(m) + ed(m,n)*Depslon(n)
  500   continue
  400 continue
c
      sigmean = sigma(1)+sigma(2)+sigma(3)/3.0d0
c
c
c
c apply plasticity model : Drucker-Prager yield criterion
c
      if(kickPL .gt. 1)
     +call PLmodel(sigma,eyng,poisson,ft,coh,phiangle,kickPL,
     +          eyngt,beta,PLalpha,PLr,sigmean,mtyp2,elplas,i,k)
c
c  transform local nodal stresses into global
c  --sig(i,k) = sigma{i=x-(1),y-(2),z-(3),xy-(4),
c                       yz-(5),xz-(6)},gauss pts(k=1,8)
c
      te(1,1)=rl1*rl1
      te(1,2)=rm1*rm1
      te(1,3)=rn1*rn1
      te(1,4)=rl1*rm1
      te(1,5)=rm1*rn1
      te(1,6)=rn1*rl1
      te(2,1)=rl2*rl2
      te(2,2)=rm2*rm2
      te(2,3)=rn2*rn2
      te(2,4)=rl2*rm2
      te(2,5)=rm2*rn2
      te(2,6)=rn2*rl2
      te(3,1)=rl3*rl3
      te(3,2)=rm3*rm3
      te(3,3)=rn3*rn3
      te(3,4)=rl3*rm3
      te(3,5)=rm3*rn3
      te(3,6)=rn3*rl3
      te(4,1)=2.0d0*rl1*rl2
      te(4,2)=2.0d0*rm1*rm2
      te(4,3)=2.0d0*rn1*rn2
      te(4,4)=rl1*rm2+rl2*rm1
      te(4,5)=rm1*rn2+rm2*rn1
      te(4,6)=rn1*rl2+rn2*rl1
```

```
      te(5,1)=2.0d0*rl2*rl3
      te(5,2)=2.0d0*rm2*rm3
      te(5,3)=2.0d0*rn2*rn3
      te(5,4)=rl2*rm3+rl3*rm2
      te(5,5)=rm2*rn3+rm3*rn2
      te(5,6)=rn2*rl3+rn3*rl2
      te(6,1)=2.0d0*rl3*rl1
      te(6,2)=2.0d0*rm3*rm1
      te(6,3)=2.0d0*rn3*rn1
      te(6,4)=rl3*rm1+rl1*rm3
      te(6,5)=rm3*rn1+rm1*rn3
      te(6,6)=rn3*rl1+rn1*rl3
c
      do 777 mi=1,6
      sig(k,mi) = 0.0d0
      do 810 ni=1,6
        sig(k,mi) = sig(k,mi) + te(ni,mi)*sigma(ni)
  810 continue
  777 continue
c
c  compute local element internal nodal forces
c
      do 600 m=1,24
      do 700 n=1,6
       f(m) = f(m) + w(k)*b(n,m)*sigma(n)*detJ
  700 continue
  600 continue
c
      PLrN((i-1)*8+k) = PLr
      do 800 m=1,6
       ny = ((i-1)*8+k-1)*6+m
       sigmaN(ny) = sigma(m)
       PLalphaN(ny) = PLalpha(m)
  800 continue
  201 continue
c
c Extrapolate stress from Gauss's pts to node
c and calculate average nodal stress
c
      call stress(sig,n1,n2,n3,n4,n5,n6,n7,n8,s1,s2,s3,s4,s5,s6,s7)

c
c  transform local internal nodal forces to global internal nodal forces
c
      detT = rl1*rm2*rn3+rl2*rm3*rn1+rl3*rm1*rn2
     +       -rl3*rm2*rn1-rl2*rm1*rn3-rl1*rm3*rn2
       sl1 =  (rm2*rn3 - rm3*rn2)/detT
       sm1 = -(rm1*rn3 - rm3*rn1)/detT
       sn1 =  (rm1*rn2 - rm2*rn1)/detT
       sl2 = -(rl2*rn3 - rl3*rn2)/detT
       sm2 =  (rl1*rn3 - rl3*rn1)/detT
       sn2 = -(rl1*rn2 - rl2*rn1)/detT
       sl3 =  (rl2*rm3 - rl3*rm2)/detT
       sm3 = -(rl1*rm3 - rl3*rm1)/detT
       sn3 =  (rl1*rm2 - rl2*rm1)/detT
c
       f1x = sl1*f(1)  + sm1*f(2)  + sn1*f(3)
       f1y = sl2*f(1)  + sm2*f(2)  + sn2*f(3)
       f1z = sl3*f(1)  + sm3*f(2)  + sn3*f(3)
       f2x = sl1*f(4)  + sm1*f(5)  + sn1*f(6)
```

```
      f2y = sl2*f(4)  + sm2*f(5)  + sn2*f(6)
      f2z = sl3*f(4)  + sm3*f(5)  + sn3*f(6)
      f3x = sl1*f(7)  + sm1*f(8)  + sn1*f(9)
      f3y = sl2*f(7)  + sm2*f(8)  + sn2*f(9)
      f3z = sl3*f(7)  + sm3*f(8)  + sn3*f(9)
      f4x = sl1*f(10) + sm1*f(11) + sn1*f(12)
      f4y = sl2*f(10) + sm2*f(11) + sn2*f(12)
      f4z = sl3*f(10) + sm3*f(11) + sn3*f(12)
      f5x = sl1*f(13) + sm1*f(14) + sn1*f(15)
      f5y = sl2*f(13) + sm2*f(14) + sn2*f(15)
      f5z = sl3*f(13) + sm3*f(14) + sn3*f(15)
      f6x = sl1*f(16) + sm1*f(17) + sn1*f(18)
      f6y = sl2*f(16) + sm2*f(17) + sn2*f(18)
      f6z = sl3*f(16) + sm3*f(17) + sn3*f(18)
      f7x = sl1*f(19) + sm1*f(20) + sn1*f(21)
      f7y = sl2*f(19) + sm2*f(20) + sn2*f(21)
      f7z = sl3*f(19) + sm3*f(20) + sn3*f(21)
      f8x = sl1*f(22) + sm1*f(23) + sn1*f(24)
      f8y = sl2*f(22) + sm2*f(23) + sn2*f(24)
      f8z = sl3*f(22) + sm3*f(23) + sn3*f(24)
c
c  assemble to global nodal force matrix --- pint (internal)
c
      pint(m1+1) = pint(m1+1) + f1x
      pint(m1+2) = pint(m1+2) + f1y
      pint(m1+3) = pint(m1+3) + f1z
      pint(m2+1) = pint(m2+1) + f2x
      pint(m2+2) = pint(m2+2) + f2y
      pint(m2+3) = pint(m2+3) + f2z
      pint(m3+1) = pint(m3+1) + f3x
      pint(m3+2) = pint(m3+2) + f3y
      pint(m3+3) = pint(m3+3) + f3z
      pint(m4+1) = pint(m4+1) + f4x
      pint(m4+2) = pint(m4+2) + f4y
      pint(m4+3) = pint(m4+3) + f4z
      pint(m5+1) = pint(m5+1) + f5x
      pint(m5+2) = pint(m5+2) + f5y
      pint(m5+3) = pint(m5+3) + f5z
      pint(m6+1) = pint(m6+1) + f6x
      pint(m6+2) = pint(m6+2) + f6y
      pint(m6+3) = pint(m6+3) + f6z
      pint(m7+1) = pint(m7+1) + f7x
      pint(m7+2) = pint(m7+2) + f7y
      pint(m7+3) = pint(m7+3) + f7z
      pint(m8+1) = pint(m8+1) + f8x
      pint(m8+2) = pint(m8+2) + f8y
      pint(m8+3) = pint(m8+3) + f8z
   13 return
      end
c
*****************************************************************************
* Drucker-Prager yield criterion with non-associate flow rule and           *
* linear combination of isotropic and kinematic hardening/softening         *
* Krieq and Key's radial-return algorithm for elastoplastic case            *
*          ( kickPL = 3 for Drucker-Prager)                                 *
*          ( beta = 0. for kinematic & =1 for isotropic hardening)          *
*****************************************************************************
c
      subroutine PLmodel(sigma,eyng,poisson,ft,coh,phiangle,kickPL,
     +            eyngt,beta,PLalpha,PLr,sigmean,mtyp2,elplas,i,k)
```

```
c
      implicit real*8 (a-h,o-z)
      dimension PLalpha(1),sigma(1),xi(6),elplas(1)
c
      Ep = eyngt/(1.0d0-eyngt/eyng)
      gshear = eyng/(2.0d0*(1.0d0+poisson))
      c1 = 2.0d0*gshear + Ep*2.0d0/3.0d0
      c2 = 2.0d0/3.0d0*beta*Ep
      c3 = 2.0d0/3.0d0*(1.0d0-beta)*Ep
      c4 = 2.0d0*gshear
c
c calculate xi-trial
c
      do 10 j=1,6
       xi(j) = sigma(j) - PLalpha(j)
   10 continue
c
c calculate deviatoric part of xi
c
      ximean = (xi(1)+xi(2)+xi(3))/3.0d0
c
c calculate the radius of cross section of cone/cylinder
c
      if ((mtyp2 .eq. 2).and.(kickPL .eq. 2)) phiangle = 0.0d0
      if(kickPL .eq. 2) then
       b = 0.0d0
       ak = ft/dsqrt(3.0d0)
      endif
      if (kickPL .eq. 3) then
       b = dsqrt(12.d0)*dsin(phiangle)/(3.0d0+dsin(phiangle))
       ak = dsqrt(12.d0)*(-coh)*dcos(phiangle)/(3.d0+dsin(phiangle))
      endif
       r = dabs(dsqrt(2.d0)*(ak+b*sigmean))
      if (r .gt. PLr) PLr = r
c
      do 20 j =1,3
       xi(j) = xi(j) - ximean
   20 continue
c
c check if elastic
c
      xxi = xi(1)*xi(1)+xi(2)*xi(2)+xi(3)*xi(3)+
     +     2.d0*(xi(4)*xi(4)+xi(5)*xi(5)+xi(6)*xi(6))
      yn = PLr*PLr
      if (xxi .le. yn) go to 100
c
c Plastic phase: calculate unit normal--N (also store in "xi")
c
      xxi = dsqrt(xxi)
      do 30 j =1,6
       xi(j) = xi(j)/xxi
   30 continue
c
c calculate lambda-tilde ("A")
c
      A = (xxi-PLr)/c1
c
c update stresses
c
      PLr = PLr + c2*A
```

```
      t1 = c3*A
      t2 = c4*A
      do 40 j=1,6
       PLalpha(j) = PLalpha(j) + t1*xi(j)
       sigma(j) = sigma(j) - t2*xi(j)
   40 continue
c
c  record the element number if plastic
c
      j = 8*(i-1)+k
      elplas(j) = dble(i) + 0.1d0*dble(k)
c

  100 return
      end
c
*****************************************************************************
*                                                                           *
*  compute element internal nodal streses(sigma) by  ---                    *
*  8-node solid isoparametric element (8-node hexahedral element)           *
*                                                                           *
*****************************************************************************
c
      subroutine stress(sig,n1,n2,n3,n4,n5,n6,n7,n8,
     +                  s1,s2,s3,s4,s5,s6,s7)
c
      implicit real*8 (a-h,o-z)
      dimension x(8),y(8),z(8)
      dimension sig(8,6),signodeX(8),signodeY(8),signodeZ(8)
      dimension signodeXY(8),signodeYZ(8),signodeXZ(8)
      dimension sN(8,8),s1(1),s2(1),s3(1),s4(1),s5(1),s6(1),s7(1)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
c
c Interpolate stresses from gauss points to nodes
c ( si(idof,io) ; idof=dof, io= local node no. )
c
      constant = dsqrt(3.0d0)
      x(1) = -constant
      y(1) = -constant
      z(1) =  constant
c
      x(2) =  constant
      y(2) = -constant
      z(2) =  constant
c
      x(3) =  constant
      y(3) =  constant
      z(3) =  constant
c
      x(4) = -constant
      y(4) =  constant
      z(4) =  constant
c
      x(5) = -constant
      y(5) = -constant
      z(5) = -constant
c
      x(6) =  constant
      y(6) = -constant
```

```
      z(6) = -constant
c
      x(7) =  constant
      y(7) =  constant
      z(7) = -constant
c
      x(8) = -constant
      y(8) =  constant
      z(8) = -constant
c
      do 203 j = 1,8
      sN(j,1) = (1.d0-x(j))*(1.d0-y(j))*(1.d0+z(j))/8.d0
      sN(j,2) = (1.d0+x(j))*(1.d0-y(j))*(1.d0+z(j))/8.d0
      sN(j,3) = (1.d0+x(j))*(1.d0+y(j))*(1.d0+z(j))/8.d0
      sN(j,4) = (1.d0-x(j))*(1.d0+y(j))*(1.d0+z(j))/8.d0
      sN(j,5) = (1.d0-x(j))*(1.d0-y(j))*(1.d0-z(j))/8.d0
      sN(j,6) = (1.d0+x(j))*(1.d0-y(j))*(1.d0-z(j))/8.d0
      sN(j,7) = (1.d0+x(j))*(1.d0+y(j))*(1.d0-z(j))/8.d0
      sN(j,8) = (1.d0-x(j))*(1.d0+y(j))*(1.d0-z(j))/8.d0
  203 continue
c
      do 501 j = 1,8
        signodeX(j) = 0.0d0
        signodeY(j) = 0.0d0
        signodeZ(j) = 0.0d0
        signodeXY(j) = 0.0d0
        signodeYZ(j) = 0.0d0
        signodeXZ(j) = 0.0d0
         do 502 k = 1,8
           signodeX(j) = signodeX(j) + sN(j,k)*sig(k,1)
           signodeY(j) = signodeY(j) + sN(j,k)*sig(k,2)
           signodeZ(j) = signodeZ(j) + sN(j,k)*sig(k,3)
           signodeXY(j) = signodeXY(j) + sN(j,k)*sig(k,4)
           signodeYZ(j) = signodeYZ(j) + sN(j,k)*sig(k,5)
           signodeXZ(j) = signodeXZ(j) + sN(j,k)*sig(k,6)
  502 continue
  501 continue
c
      s1(n1) = s1(n1) + signodeX(1)
      s1(n2) = s1(n2) + signodeX(2)
      s1(n3) = s1(n3) + signodeX(3)
      s1(n4) = s1(n4) + signodeX(4)
      s1(n5) = s1(n5) + signodeX(5)
      s1(n6) = s1(n6) + signodeX(6)
      s1(n7) = s1(n7) + signodeX(7)
      s1(n8) = s1(n8) + signodeX(8)

c
      s2(n1) = s2(n1) + signodeY(1)
      s2(n2) = s2(n2) + signodeY(2)
      s2(n3) = s2(n3) + signodeY(3)
      s2(n4) = s2(n4) + signodeY(4)
      s2(n5) = s2(n5) + signodeY(5)
      s2(n6) = s2(n6) + signodeY(6)
      s2(n7) = s2(n7) + signodeY(7)
      s2(n8) = s2(n8) + signodeY(8)

c
      s3(n1) = s3(n1) + signodeZ(1)
      s3(n2) = s3(n2) + signodeZ(2)
```

```
      s3(n3) = s3(n3) + signodeZ(3)
      s3(n4) = s3(n4) + signodeZ(4)
      s3(n5) = s3(n5) + signodeZ(5)
      s3(n6) = s3(n6) + signodeZ(6)
      s3(n7) = s3(n7) + signodeZ(7)
      s3(n8) = s3(n8) + signodeZ(8)
c
      s4(n1) = s4(n1) + signodeXY(1)
      s4(n2) = s4(n2) + signodeXY(2)
      s4(n3) = s4(n3) + signodeXY(3)
      s4(n4) = s4(n4) + signodeXY(4)
      s4(n5) = s4(n5) + signodeXY(5)
      s4(n6) = s4(n6) + signodeXY(6)
      s4(n7) = s4(n7) + signodeXY(7)
      s4(n8) = s4(n8) + signodeXY(8)
c
      s5(n1) = s5(n1) + signodeYZ(1)
      s5(n2) = s5(n2) + signodeYZ(2)
      s5(n3) = s5(n3) + signodeYZ(3)
      s5(n4) = s5(n4) + signodeYZ(4)
      s5(n5) = s5(n5) + signodeYZ(5)
      s5(n6) = s5(n6) + signodeYZ(6)
      s5(n7) = s5(n7) + signodeYZ(7)
      s5(n8) = s5(n8) + signodeYZ(8)
c
      s6(n1) = s6(n1) + signodeXZ(1)
      s6(n2) = s6(n2) + signodeXZ(2)
      s6(n3) = s6(n3) + signodeXZ(3)
      s6(n4) = s6(n4) + signodeXZ(4)
      s6(n5) = s6(n5) + signodeXZ(5)
      s6(n6) = s6(n6) + signodeXZ(6)
      s6(n7) = s6(n7) + signodeXZ(7)
      s6(n8) = s6(n8) + signodeXZ(8)
c
      s7(n1) = s7(n1) + 1.0d0
      s7(n2) = s7(n2) + 1.0d0
      s7(n3) = s7(n3) + 1.0d0
      s7(n4) = s7(n4) + 1.0d0
      s7(n5) = s7(n5) + 1.0d0
      s7(n6) = s7(n6) + 1.0d0
      s7(n7) = s7(n7) + 1.0d0
      s7(n8) = s7(n8) + 1.0d0
c
   13 return
      end
*************************************************************************
*                                                                       *
*  print out final results requested at specific points & directions    *
*                                                                       *
*************************************************************************
c
      subroutine prout (numout,rkout,d,v,a,s1,s2,s3,s4,s5,s6,s7,ndof,
     +                  nstep,time,proutv)
c
c  This subroutine controls output of displacement, velocity,
c  acceleration, and stresses.
c
      implicit real*8 (a-h,o-z)
      dimension d(ndof,1),v(ndof,1),a(ndof,1)
      dimension rkout(3,1),proutv(1),s1(1),s2(1),s3(1),s4(1)
```

```
      dimension s5(1),s6(1),s7(1)
c
      do 100 i2=1,numout
      node = int(rkout(1,i2))
      idva = int(rkout(2,i2))
      idof = int(rkout(3,i2))
       if (idva .eq. 0) proutv(i2) = d(idof,node)
       if (idva .eq. 1) proutv(i2) = v(idof,node)
       if (idva .eq. 2) proutv(i2) = a(idof,node)
       if (idva .eq. 3) then
        if (idof .eq. 1) proutv(i2) = s1(node)/s7(node)
        if (idof .eq. 2) proutv(i2) = s2(node)/s7(node)
        if (idof .eq. 3) proutv(i2) = s3(node)/s7(node)
        if (idof .eq. 4) proutv(i2) = s4(node)/s7(node)
        if (idof .eq. 5) proutv(i2) = s5(node)/s7(node)
        if (idof .eq. 6) proutv(i2) = s6(node)/s7(node)
       end if
  100 continue
       write (*,200) nstep
  200 format(' [',i7,']')
      write(7,300) time,(proutv(i),i=1,numout)
  300 format(1x,'time =',e12.5,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/19x,6(1x,e9.3)/
     +      19x,6(1x,e9.3))
c
      return
      end
c
*********************************************************************
*                                                                   *
*  read in node & element data , material properties                *
*     acc. & force data, initial condition data, and output request *
*                                                                   *
*********************************************************************
c
      subroutine readata (nnd,nel,nummat,numout,iacc,ndof,rnode,
     +                    iforce,imesh,xc,yc,zc,rifix,e,rkout,inital)
c
      implicit real*8 (a-h,o-z)
      dimension rifix(1),rnode(12,1),rkout(3,1),xc(1),yc(1),zc(1)
      dimension ifixx(3),node(12),kout(3),e(11,1)
c
      common /box 1/ iprob,delta,alpha,toler,gravity
      common /box 4/ npnts,numif,nnaf,ndisi,nveli,
     +               kfpnts(6),jnode(30),jdir(30),jaf(30),
     +               ndnod(10),kdis(10),nvnod(10),kvel(10)
      common /box 4a/g,disi(10),veli(10),f(10),omega(10),
     +               ff(500,6),ta(500,6),tf(500,6),aa(500,6)
```

```
c
      meq = nnd*ndof
c
      do 100 i=1,meq
  100 rifix(i) = 0.0
c
c  read & write nodal coord. &  B.C.
c  skipped nodes will be automatically generated
c
      write(6,550)
      l = 0
  110 read(5,*)  n,xc(n),yc(n),zc(n),(ifixx(i),i=1,3)
c
      do 120 j=1,3
      n1 = (n-1)*ndof
      rifix(n1+j) = dble(ifixx(j))
  120 continue
c
      n1 = l+1
c
      if (l .eq. 0) go to 130
c
      dl = n-l
      dxl = (xc(n)-xc(l))/dl
      dyl = (yc(n)-yc(l))/dl
      dzl = (zc(n)-zc(l))/dl
c
  130 continue
      l = l+1

      if (n-l) 180,170,150
c
  150 xc(l) = xc(l-1)+dxl
      yc(l) = yc(l-1)+dyl
      zc(l) = zc(l-1)+dzl
c
      do 155 j=1,3
      l1 = (l-1)*ndof
      l2 = (l-2)*ndof
      rifix(l1+j) = rifix(l2+j)
  155 continue
c
      if (imesh .eq. 1) then
       write(6,570) l,xc(l),yc(l),zc(l),(int(rifix(i)),i=l1+1,l1+3)
      endif
      go to 130
c
 170  continue
      if (imesh .eq. 1) then
       write(6,570) n,xc(n),yc(n),zc(n),(int(rifix(k)),k=n1+1,n1+3)
      endif
      if (nnd-n) 180,190,110
c
  180 continue
      write(6,580) n
      stop
c
  190 continue
c
```

```
c  read & write element connectivity
c
      write(6,600)
      l = 0
  200 read(5,*) n,node(1),node(2),node(3),node(4),node(5),node(6),
     +         node(7),node(8),node(9),node(10),node(11),node(12),knt
c
      do 212 i=1,12
      rnode(i,n) = dble(node(i))
  212 continue
c
      nl = l+1
      if (knt .eq. 0) knt = 1
  245 continue
      l = l+1
      if (n-l) 260,255,250
 250  rnode(1,l)  = rnode(1,l-1) + dble(knt)
      rnode(2,l)  = rnode(2,l-1) + dble(knt)
      rnode(3,l)  = rnode(3,l-1) + dble(knt)
      rnode(4,l)  = rnode(4,l-1) + dble(knt)
      rnode(5,l)  = rnode(5,l-1) + dble(knt)
      rnode(6,l)  = rnode(6,l-1) + dble(knt)
      rnode(7,l)  = rnode(7,l-1) + dble(knt)
      rnode(8,l)  = rnode(8,l-1) + dble(knt)
      rnode(9,l)  = rnode(9,l-1)
      rnode(10,l) = rnode(10,l-1)
      rnode(11,l) = rnode(11,l-1) + dble(knt)
      rnode(12,l) = rnode(12,l-1)
      go to 245
  255 continue
      if (imesh .eq. 1) then
      write(6,620)(k,int(rnode(1,k)),int(rnode(2,k)),int(rnode(3,k)),
     +               int(rnode(4,k)),int(rnode(5,k)),int(rnode(6,k)),
     +               int(rnode(7,k)),int(rnode(8,k)),int(rnode(9,k)),
     +               int(rnode(10,k)),int(rnode(11,k)),int(rnode(12,k)),
     +               k = nl,n)
      endif
      if (nel-n) 260,270,200
c
  260 continue
      write(6,630) n
      stop
  270 continue
c
c  read & write material properties
c
      do 300 i=1,nummat
      read(5,*)  k,e(1,k),e(2,k),e(3,k),e(4,k),e(5,k)
      read(5,*)  e(6,k),e(7,k),e(8,k),e(9,k),e(10,k),e(11,k)
      write(6,960)
      write(6,970) k,int(e(1,k)),e(2,k),e(3,k),e(4,k),e(5,k)
      write(6,980)
      write(6,990) e(6,k),e(7,k),int(e(8,k)),e(9,k),e(10,k)
  300 continue
c
c  read & write no. of acceleration history, and data of time-acc. pairs
c  in each acc. history
c
      if (iacc .eq. 0) go to 760
c
```

```
      read(5,*)    nacc,npnts
      read(5,*)    g
      write(6,1170)
      write(6,1180) nacc,npnts
c
      do 721 i=1,nacc
      read(5,*)  (ta(j,i),aa(j,i),j=1,npnts)
  721 continue
c
  760 continue
      if (iforce .eq. 0) go to 762
      if (iforce .ne. 1) go to 763
c
c  read & write no. of impact force history, no. of nodes, and data of
c  time-acc. pairs in each acc. history
c
      read(5,*)     numif,nnaf
      write(6,1300)
      write(6,1310) numif,nnaf
c
      if (numif .eq. 0) go to 768
c
      do 755 i=1,numif
      read(5,*)  kfpnts(i)
      jf = kfpnts(i)
      read(5,*)  (tf(j,i),ff(j,i),j=1,jf)
      write(6,1295)
      do 785 j=1,jf
      write(6,1210) i,j,tf(j,i),ff(j,i)
  785 continue
  755 continue
c
  768 continue
      write(6,1315)
c
c  read & write data of position applied by arbitrary shape impact force
c  function ( node, d.o.f., history no. )
c
      do 769 i=1,nnaf
      read(5,*)    jnode(i),jdir(i),jaf(i)
      write(6,1225)jnode(i),jdir(i),jaf(i)
  769 continue
      go to 762
c
c  read & write data of position applied by sinusoidal impact force
c  function ( node, d.o.f., history no. )
c
  763 continue
      read(5,*)  nnaf
c
      do 764 i=1,nnaf
      read(5,*)  jnode(i),jdir(i),f(i),omega(i)
  764 continue
c
  762 continue
      if (inital .eq. 0) go to 765
      if (inital .eq. 2) go to 782
c
c  read & write data of displacement type I.C. ( node, dof, I.C.value )
c
```

```
      read(5,*)  ndisi
      write(6,784)
      do 786 i=1,ndisi
      read(5,*)    ndnod(i),kdis(i),disi(i)
      write(6,783) ndnod(i),kdis(i),disi(i)
  786 continue
c
      if (inital .ne. 3) go to 765
c
c  read & write data of velocity type I.C. ( node, dof, I.C.value )
c
  782 read(5,*)  nveli
      write(6,787)
      do 788 i=1,nveli
      read(5,*)    nvnod(i),kvel(i),veli(i)
      write(6,783) nvnod(i),kvel(i),veli(i)
  788 continue
c
c  read & write data of output record requested (node, d-v-a-type, dof)
c
  765 continue

      if (numout .eq. 0) go to 780
c
      write(6,1228)
      write(7,1228)
c
      do 770 i=1,numout
       read(5,*) (kout(j),j=1,3)
       write(6,1229) i,(int(kout(j)),j=1,3)
       write(7,1229) i,(int(kout(j)),j=1,3)
      do 771 j=1,3
       rkout(j,i) = dble(kout(j))
  771 continue
  770 continue
  780 continue
c
  550 format (/2x,'card  4',7x,'nodal  point data',/,
     +        7x,'node no.',3x,'x-ord',4x,'y-ord',4x,'z-ord'
     +        6x,'ifx',5x,'ify',6x,'ifz')
  570 format (10x,i4,2x,f6.2,2x,f6.2,5x,f6.2,3i8)
  580 format (17x,'nodal point error n =',i5)
  600 format(/2x,'card  5     element data'/,1x,'ele.no.',3x,'N1',
     +        3x,'N2',3x,'N3',3x,'N4',3x,'N5',3x,'N6',3x,'N7',3x,'N8',
     +        2x,'mat',2x,'row',2x,'col',1x,'E-con.')
  620 format (1x,i4,4x,i4,1x,i4,1x,i4,1x,i4,1x,i4,1x,i4,1x,i4,1x,
     +        i4,1x,i4,2x,i2,1x,i4,1x,i4,4x,i1)
  630 format (17x,'element number error n =',i5)
c
  783 format (i8,6x,i6,10x,f12.5)
  784 format (5x,'node',5x,'disl-dir',8x,'initial disp',/)
  787 format (5x,'node',5x,'vel-dir',8x,'initial vel',/)
c
  960 format (/2x,'card  6 & 7    material property data',/,10x,
     +'material  material  mass      Youngs    Poisson   tensile'/10x,
     +'group no. type no.  density   modulus    ratio     strength')
  970 format (11x,i3,7x,i3,3x,e10.4,1x,e10.4,2x,f5.3,3x,e10.4)
  980 format (19x,'cohesion   phi     yield     tangent  hardening',/30x
     +'angle   criterion  modulus    rule')
  990 format (17x,e10.4,2x,f6.2,5x,i2,5x,e9.4,1x,f5.3)
```

```
 1170 format (/,2x,'card  12',6x,'prescribed acceleration card'/)
 1180 format (10x,'total no. of acceleration histories                =',
     +   i5,/,10x,'total no. of time-acc. pairs in each acc. history =',
     +   i5)
 1200 format (/,2x,'card  13',6x,'acceleration-history card',/,
     +        17x,'pair no.',8x,'time',9x,'acc'/)
 1210 format (20x,i6,11x,i6,4x,e12.4,4x,e12.4)
 1220 format (/,2x,'card  14',6x,'nodal accele. information card',/,
     +        17x,' node','  dir','  acc'/)
 1225 format (15x,i5,8x,i5,14x,i5)
 1228 format (/,2x,'card  21   output information card',/,14x,
     + 'seq.    node#      d-(0),v-(1),a-(2),   x(1),y(2),z(3)',/,38x,
     + 'stress-(3)',6x,'xy(4),yz(5),xz(6)')
 1229 format (12x,i4,6x,i4,13x,i4,16x,i4)
 1295 format (/2x,'card  12 & 13     impact force history card',/,17x,
     +       'force history no.'5x,'pair no.',5x,'time',9x,'iforce')
 1300 format (/2x,'card  11     prescribed impact force')
 1310 format (20x,'total no. of impact force history          =',i5,/,
     +        20x,'total no. of nodes applied by impact force =',i5)
 1315 format (/2x,'card  14     nodal impact force information',/,
     +        15x,'node no.    x-(1),y-(2),z-(3)   force history no.')
c
      return
      end
c
c
**************************************************************************
*                                                                        *
* compute the rigid body rotation by cross product between               *
*            undeformed and deformed vectors                             *
*                                                                        *
*                                                                        *
**************************************************************************
c
      subroutine thetafind (xcd1,ycd1,zcd1,xcd2,ycd2,zcd2,
     +                      xu,yu,zu,xd,yd,zd,thetaX,thetaY,thetaZ)
c
      implicit real*8 (a-h,o-z)
      data phi/3.1415926535898d0/
c
      xau = xu - xcd1
      yau = yu - ycd1
      zau = zu - zcd1
c
      xad = xd - xcd2
      yad = yd - ycd2
      zad = zd - zcd2
c
c compute the length of undeformed shape
c
      rLuxy = dsqrt(xau*xau + yau*yau)
      rLuyz = dsqrt(yau*yau + zau*zau)
      rLuxz = dsqrt(xau*xau + zau*zau)
c
c compute the length of deformed shape
c
      rLdxy = dsqrt(xad*xad + yad*yad)
      rLdyz = dsqrt(yad*yad + zad*zad)
      rLdxz = dsqrt(xad*xad + zad*zad)
c
```

```
c compute the unit components of undeformed vectors
c
c xy plane = 1
      eux1 = xau/rLuxy
      euy1 = yau/rLuxy
c yz plane = 2
      euy2 = yau/rLuyz
      euz2 = zau/rLuyz
c xz plane = 3
      eux3 = xau/rLuxz
      euz3 = zau/rLuxz
c
c compute the unit components of deformed vectors
c
c xy plane = 1
      edx1 = xad/rLdxy
      edy1 = yad/rLdxy
c yz plane = 2
      edy2 = yad/rLdyz
      edz2 = zad/rLdyz
c xz plane = 3
      edx3 = xad/rLdxz
      edz3 = zad/rLdxz
c
c compute the rigid body rotation
c
c xy plane rotation(thetaZ)
      Cz = eux1*edy1 - edx1*euy1
      thetaZ = dasin(Cz)
      if ((xu*xd .lt. 0.d0).and.(yu*yd .lt. 0.d0)) then
       if (thetaZ .ge. 0.d0) thetaZ =  phi - thetaZ
       if (thetaZ .lt. 0.d0) thetaZ = -phi - thetaZ
      endif
c yz plane rotation(thetaX)
      Cx = euy2*edz2 - edy2*euz2
      thetaX = dasin(Cx)
      if ((yu*yd .lt. 0.d0).and.(zu*zd .lt. 0.d0)) then
       if (thetaX .ge. 0.d0) thetaX =  phi - thetaX
       if (thetaX .lt. 0.d0) thetaX = -phi - thetaX
      endif
c xz plane rotation(thetaX)
      Cy = euy2*edz2 - edy2*euz2
      thetaY = dasin(Cy)
      if ((xu*xd .lt. 0.0d0).and.(zu*zd .lt. 0.0d0)) then
       if (thetaY .ge. 0.d0) thetaX =  phi - thetaY
       if (thetaY .lt. 0.d0) thetaX = -phi - thetaY
      endif
c
      return
      end
c
```

Appendix 2

Verification of the three dimensional finite element code

Problem 1. A rectangular plate (or half space) of elastic material subjected to ramp loadings. Deflections are compared with the solutions obtained by using ANSYS.

Problem 2. A rectangular plate (or half space) of elastic-plastic material subjected to ramp loadings (Mises criterion and associated flow rule). Progress of the plastic zone are shown. Deflections are compared with the solutions obtained by using ANSYS.

Problem 3. A rectangular plate (or half space) of elastic-plastic material subjected to ramp loadings (Drucker-Prager criterion and associated flow rule). Progress of the plastic zone are shown. Deflections are compared with the solutions obtained by using ANSYS.

Problem 4. A rectangular plate of elastic-plastic material with Mises criterion subjected to sinusoidal loadings (Response is in the elastic range).

Problem 5. A rectangular plate of elastic-plastic material with Mises criterion subjected to pulse loadings (Response is in the elastic range).

Problem 6. A rectangular plate of viscoelastic material of Maxwell type subjected to ramp loadings